Tampereen yliopisto
Tampere University

COMP.SE.100-EN ItSE

Zoom begins soon...

at 1415 o'clock.

Tampereen yliopisto
Tampere University

COMP.SE.100-EN, 2020, course schedule v6d (21.10.2020)

week	lectures	exam	weekly exercises	project assignment (exercise work)	week
35	L1: course basics		--- sign to WE groups ---	sign for project = grouping...	35
36	Project Assignment explained		**WE1**: intro to requirements	grouping, groups to Moodle	36
37	L2: Sw Eng in general		**WE2**: Trello and agile way	group's Trello board ready with product backlog	37
38	L3: requirements		**WE3**: feasibility study and stakeholder analysis	working...	38
39	L4: basic UML diagrams		**WE4**: requirements	working...	39
40	L5: more UML diagrams	EXAM-1	**WE5**: UML diagrams - Use case	working...	40
41	L6: different sw systems	EXAM-1	**WE6**: UML diagrams - concept/entity and navigation	deadline for 1st phase documentation and presentation	41
42	examination week		examination week	examination week	42
43	L7: life cycle models		groups' 1st presentations	groups' 1st phase presentations	43
44	L8: quality and testing		**WE7**: development processes	feedback group-to-group at PRP, from 1st phase	44
45	L9: project work	Forms-2	**WE8**: testing and error reporting	deadline for diagrams first versions (Moodle)	45
46	L10: project management		**WE9**: effort estimation	feedback to groups from diagrams (from assistants)	46
47	L11: open source, APIs, IPR		**WE10**: delivery contracts and terms of use	deadline for 2nd phase presentation (PRP)	47
48	L12: embedded systems, IoT	EXAM-3	groups' final presentations	groups' final presentations / feedback g-to-g (PRP)	48
49	L13: recap, summary	EXAM-3	- - -	final (2.) delivery of project documentation	49
50	examination week		examination week	feedback inside group, student-to-student at PRP	50
51	examination week		examination week	end of game / game over.	51
	Lectures: Wed at 1415-16.		Weekly exercises:		
			Mon 0815-10 discontinued	AUTUMN 2020 (1-2. periods)	
			Mon 1215-14	are remote/distant learning.	
			Tue 0815-10		
			Tue 1415-16		
			Wed 0815-10 discontinued .		

Remote/distant learning 2020.
No contact teaching at ItSE 2020.

Tampereen yliopisto
Tampere University

COMP.SE.100 -EN "ItSE"

Introduction to Software Engineering

2020, 1-2. periods

5 credit units

08-QAtest-ItSE-2020-v7

Tampereen yliopisto
Tampere University

COMP.SE.100-EN (ItSE)
Introduction to Software Engneering

Lecture 8, 28.10.2020

Tensu: remember to start Zoom lecture recording, at 1415

Prefer course Moodle over SISU information.

Students are recommended to follow Moodle News/messages.

Digital security week (in Finnish)

Course contents (plan)

1. Course basics, intro
2. Sw Eng in general, overview
3. Requirements
4. Different software systems
5. Basic UML Diagrams (”Class”, Use Case, Navigation)
6. UML diagrams, in more detail
7. Life Cycle models
8. **Quality and Testing**
9. Project work
10. Project management
11. Open source, APIs, IPR
12. Embedded systems
13. Recap

Tampereen yliopisto
Tampere University

8. Quality and Testing

- GSD = global sw dev, distributed sw dev, outsourcing, off-shoring
- maintenance
- Quality Assurance (QA)
- testing
  - review / inspection
  - unit testing
  - integration testing
  - system testing
  - usability testing
  - acceptance testing
- TDD (Test-Driven Development)
- version control
- verification and validation (V&V)
- configuration management.

Tampereen yliopisto
Tampere University

Current at course (w 44)

- WE7 were this week (Agile game)
- next week WE8 (Testing and quality)

- continue updating your Trello (kanban) boards = use at your process

- feedback group-to-group at PRP, from 1st presentations, deadline at week 45, that is 01.11.2020

- remember Forms-2 (diagrams) exam on week 45, Wed 04.11.2020 at 1615-, three questions = three diagrams, some diagramming tool must be used (not PowerPoint nor Paint)

Tampereen yliopisto
Tampere University

Some annual DAYs...

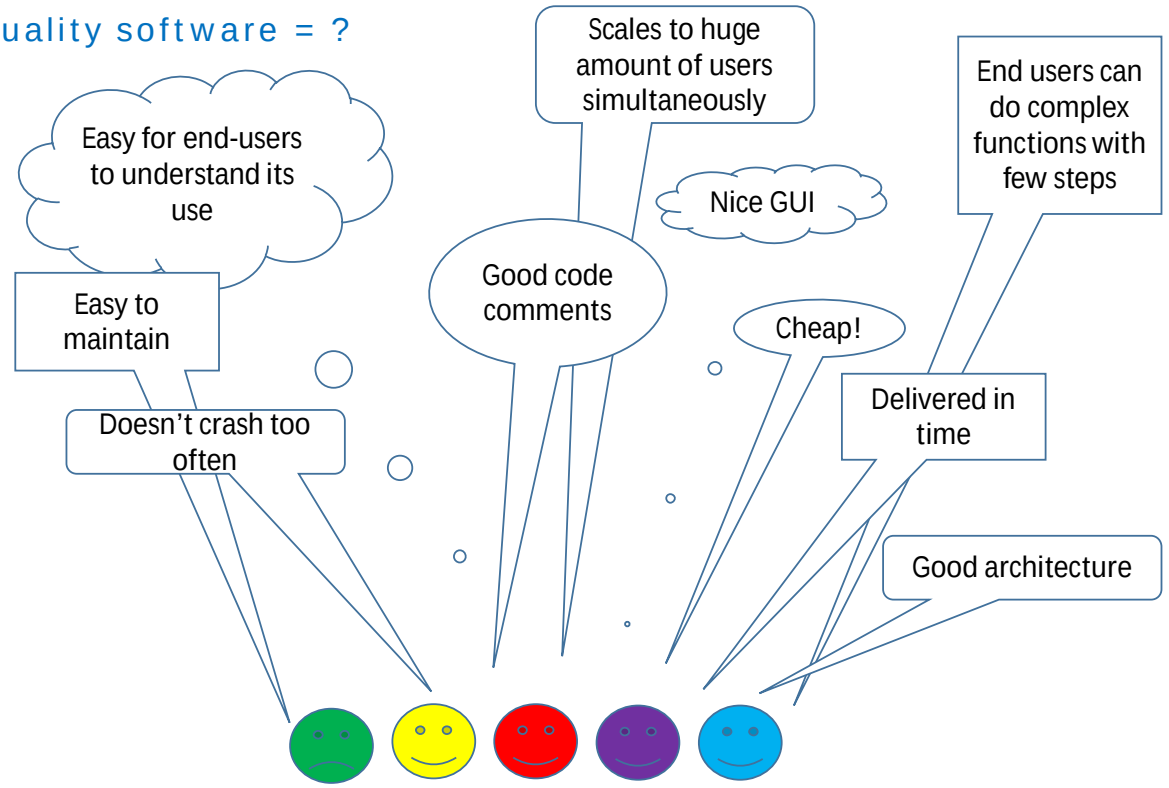
International Programmer's Day 12.09.2020
Software Freedom Day 19.09.2020

Ongoing: National Digital security week 26-30.10.2020

International Project Management Day 05.11.2020
World Usability Day 12.11.2020
World Quality Day 12.11.2020

Tampereen yliopisto
Tampere University

Quality software = ?



Tampereen yliopisto
Tampere University

QA = quality assurance, ongoing process

3.3260 , quality assurance (QA)

1. part of quality management focused on providing confidence that quality requirements will be fulfilled

2. all the planned and systematic activities implemented within the quality system, and demonstrated as needed, to provide adequate confidence that an entity will fulfill requirements for quality

Note 1 to entry: [ISO 9000:2005] There are both internal and external purposes for quality assurance: within an organization, quality assurance provides confidence to management; in contractual situations, quality assurance provides confidence to the customer or others. Some quality control and quality assurance actions are interrelated. Unless requirements for quality fully reflect the needs of the user, quality assurance does not necessarily provide adequate confidence.

3.3265 , quality control (QC)

1. set of activities designed to evaluate the quality of developed or manufactured products

2. monitoring service performance or product quality, recording results, and recommending necessary changes

cf. quality assurance

Note 1 to entry: This term has no standardized meaning in software engineering at this time.

[ISO/IEC/IEEE 24765:2017]

Tampereen yliopisto
Tampere University

GSD = Global software development

Tampereen yliopisto
Tampere University

Global, distributed,...

- GSD = global sw development; development teams in different countries, ideally 24 hours development (around-the-clock) in a day by different time-zones, BUT a lot of risks (culture, process, management,...)
- distributed sw development (multi-site); development teams are not located in the same place
- outsourcing; some other company does the work (FI: ulkoistaminen)
- subcontracting; some other company does (part of) the work (FI: alihankinta)
- off-shoring; work is done at another country (and near-shoring)
  - main reasons for off-shoring; get cheap talented work (but does it happen ?)
- reshoring; getting work back home country from abroad
- main reasons for reshoring; delivery time, total cost, quality.

The main problem: is the Sw Dev process understood the same way everywhere ? There are very few success stories.

Open Source projects may use **crowdsourcing** (FI: joukkoistaminen) .

3.1240

distributed computing

1. spreading of computation and data across a number of computers connected by a network.

[ISO 24765:2017]

It is said that communication becomes more difficult by factor 10, when developers are located at different

- workrooms
- floors
- buildings
- towns / cities
- countries.

Why outsource software development?

1 Access to a larger talent pool and the latest technology

2 Increased Focus on Core Business

3 Cost savings

4 Better Risk Management

5 Accommodate peak loads

6 Better Security

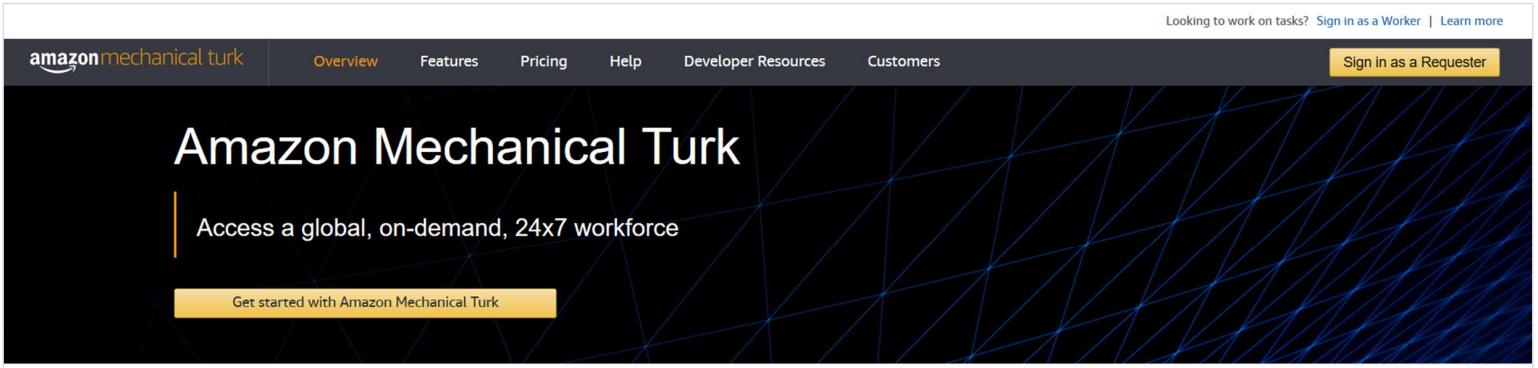
7 Spend Less Time on Support

8 Reduce time to market.

[https://hackernoon.com/8-reasons-why-outsourcing-software-development-works-f399fcb5d2d2]

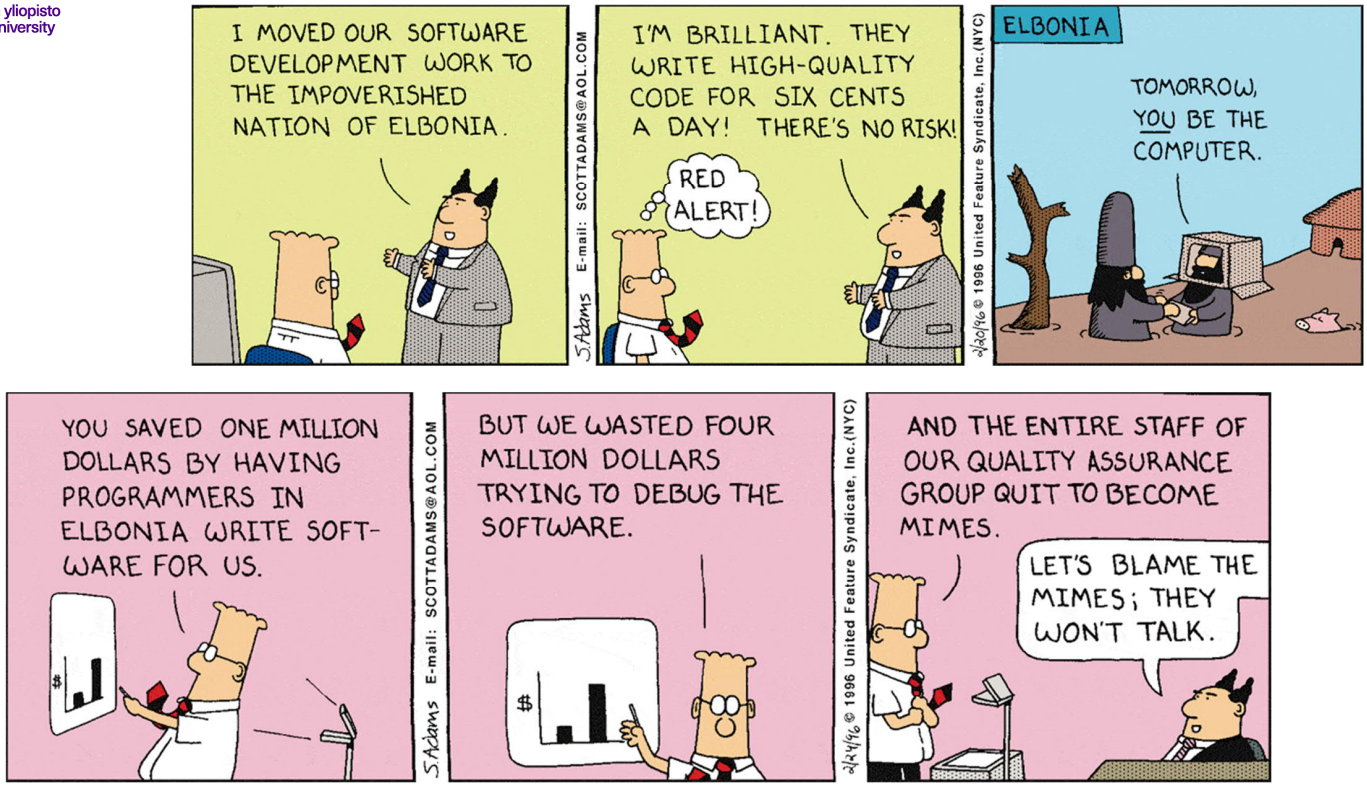
Actually there seems to be some outsourcing success stories:
http://www.invimatic.com/how-to-select-a-software-development-outsourcing-company/
https://www.daxx.com/blog/development-trends/outsourcing-success-stories
https://medium.com/@xipetechnology/software-development-outsourcing-success-stories-c0972476245c

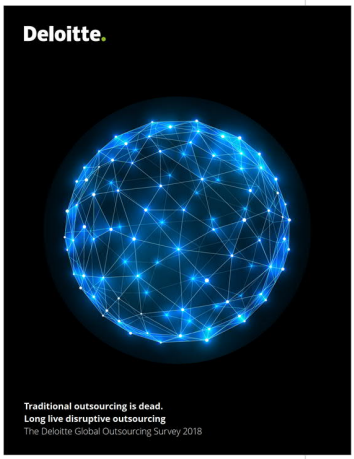
[https://perfectial.com/blog/outsourcing-software-development/]

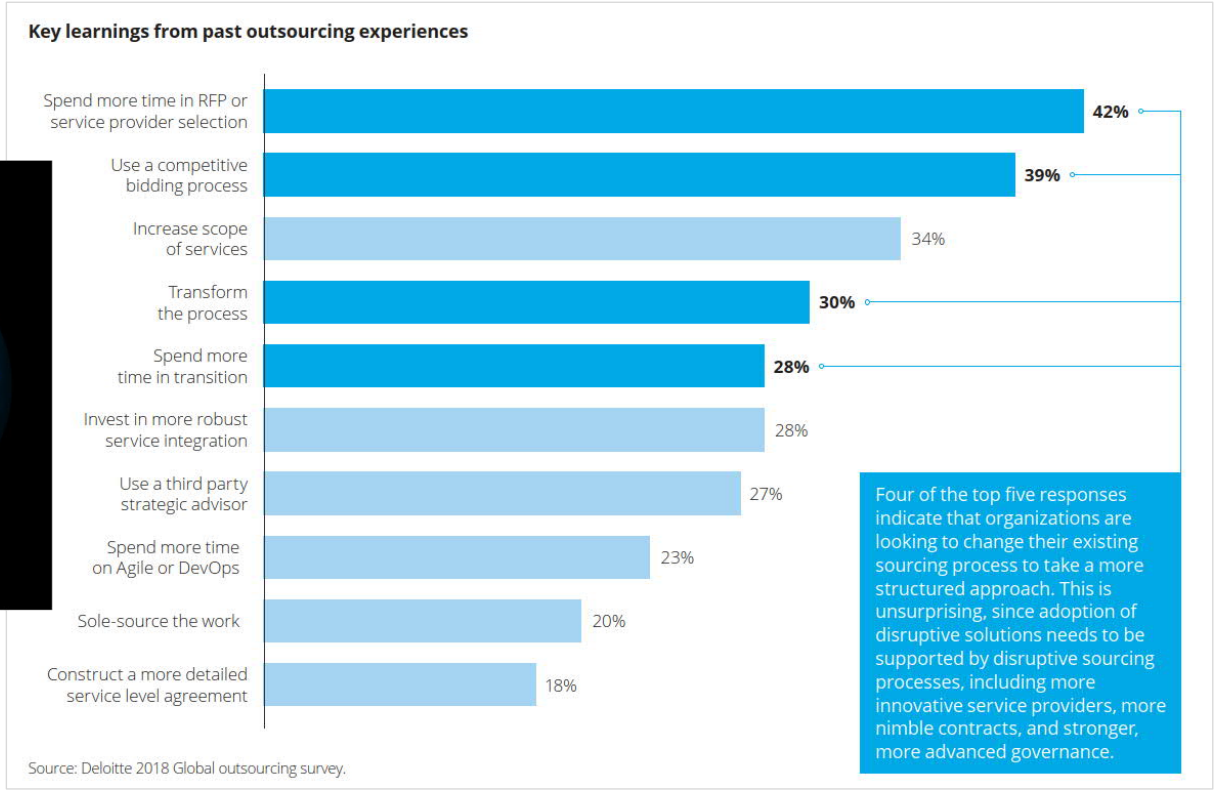


Amazon Mechanical Turk (MTurk) is a crowdsourcing marketplace that makes it easier for individuals and businesses to outsource their processes and jobs to a distributed workforce who can perform these tasks virtually. This could include anything from conducting simple data validation and research to more subjective tasks like survey participation, content moderation, and more. MTurk enables companies to harness the collective intelligence, skills, and insights from a global workforce to streamline business processes, augment data collection and analysis, and accelerate machine learning development.

While technology continues to improve, there are still many things that human beings can do much more effectively than computers, such as moderating content, performing data deduplication, or research. Traditionally, tasks like this have been accomplished by hiring a large temporary workforce, which is time consuming, expensive and difficult to scale, or have gone undone. Crowdsourcing is a good way to break down a manual, time-consuming project into smaller, more manageable tasks to be completed by distributed workers over the Internet (also known as 'microtasks').


I MOVED OUR SOFTWARE DEVELOPMENT WORK TO THE IMPOVERISHED NATION OF ELBONIA.
I'M BRILLIANT. THEY WRITE HIGH-QUALITY CODE FOR SIX CENTS A DAY! THERE'S NO RISK!
RED ALERT!
E-mail: SCOTTADAMS@AOL.COM
ELBONIA
TOMORROW, YOU BE THE COMPUTER.
© 1996 United Feature Syndicate, Inc.(NYC)
YOU SAVED ONE MILLION DOLLARS BY HAVING PROGRAMMERS IN ELBONIA WRITE SOFTWARE FOR US.
BUT WE WASTED FOUR MILLION DOLLARS TRYING TO DEBUG THE SOFTWARE.
AND THE ENTIRE STAFF OF OUR QUALITY ASSURANCE GROUP QUIT TO BECOME MIMES.
LET'S BLAME THE MIMES; THEY WON'T TALK.


Deloitte.
Traditional outsourcing is dead.
Long live disruptive outsourcing
The Deloitte Global Outsourcing Survey 2018


Key learnings from past outsourcing experiences
Spend more time in RFP or service provider selection 42%
Use a competitive bidding process 39%
Increase scope of services 34%
Transform the process 30%
Spend more time in transition 28%
Invest in more robust service integration 28%
Use a third party strategic advisor 27%
Spend more time on Agile or DevOps 23%
Sole-source the work 20%
Construct a more detailed service level agreement 18%
Four of the top five responses indicate that organizations are looking to change their existing sourcing process to take a more structured approach. This is unsurprising, since adoption of disruptive solutions needs to be supported by disruptive sourcing processes, including more innovative service providers, more nimble contracts, and stronger, more advanced governance.
Source: Deloitte 2018 Global outsourcing survey.

Tampereen yliopisto
Tampere University

Three key trends emerged from 2018's outsourcing survey

Outsourcing is enabling competitive advantage

While cost optimization is still a critically important criterion for outsourcing, it is no longer at the top of the list (nor even in the top five), since disruptive outsourcing, when executed well, can deliver competitive advantage by transforming the way organizations operate, and making them more agile, efficient, and effective. The advantages are obvious to respondents: approximately 84 percent of them have either initiated discussions, conducted pilots, or have implemented at least some disruptive solutions.

Organizations are embracing disruptive outsourcing technologies such as cloud and robotic process automation (RPA)

The vast majority of organizations—93 percent—are considering or adopting cloud solutions and 72 percent are considering or adopting RPA solutions. Seventy percent of respondents believe their service providers have a reasonable or advanced ability to implement disruptive solutions.

There are real challenges with adopting disruptive solutions

Data migration, security requirements, and application optimization/change are just a few examples of challenges related to cloud adoption. Organizational resistance, highly fragmented processes, and regulatory restraints are common challenges related to RPA adoption.

[www.deloitte.com , 2018]

Tampereen yliopisto
Tampere University

Outsourcing...

We asked respondents what they would do differently when launching their next outsourcing initiative based on their past experiences.

Service provider selection. The top responses were related to the selection process: spend more time in RFP or service provider selection (42 percent), and use a competitive bidding process (39 percent). This may be due to the increasing maturity of both the procurement and vendor management functions within organizations. Many clients use a sole source approach to service provider selection, likely with the expectation that it is faster to execute the process with a single service provider; however, they will likely pay the price through higher fees, lower service levels, and less favorable terms. And, counterintuitively, it will usually take longer, since a competitive process creates more sense of urgency than a sole source approach does (though, of course, a poor deal can always be done quite quickly).

Strategic planning approach. Other popular answers involved taking a more strategic approach and planning a new outsourcing initiative: increase the scope of service (34 percent); transform the process rather than simply lifting and shifting (30 percent); invest in more robust service integration and transition (28 percent); use a third-party advisor (27 percent).

[www.deloitte.com , 2018]

Tampereen yliopisto
Tampere University

Where to outsource ?

10 Best Countries to Outsource Software Development, Based on Data by Dianna Gunn / Updated: September 24, 2019 / [www.codeinwp.com/blog/best-countries-to-outsource-software-development/]

1. India
2. Ukraine
3. China
4. Poland
5. The Philippines
6. Romania
7. Brazil
8. Taiwan
9. Egypt
10. Canada.

Well... a Finnish consultant said some years ago, that South Africa will be the next “hot spot”, after salaries in Estonia, Russia, India and China have raised too much.

Tampereen yliopisto
Tampere University

Maintenance

Maintenance

Every commercial software has some kind of maintenance during its life cycle. Some kind of documentation need to exist to support maintenance.

Maintenance can be

- fixing
- adding
- removing
- modifying
- modernising
- supportive.

It would be good to keep maintenance in mind from the beginning of development, if you will be having maintenance in-house. Maintenance plan would be good. Coding style guide requires e.g. extensive commenting.

ISO 24765:2017 Systems and software engineering — Vocabulary

3.2314 , maintainability

1. ease with which a software system or component can be modified to change or add capabilities, correct faults or defects, improve performance or other attributes, or adapt to a changed environment

2. ease with which a hardware system or component can be retained in, or restored to, a state in which it can perform its required functions

3. capability of the software product to be modified

4. average effort required to locate and fix a software failure

5. speed and ease with which a program can be corrected or changed

6. degree of effectiveness and efficiency with which a product or system can be modified by the intended maintainers

cf. extendability, flexibility

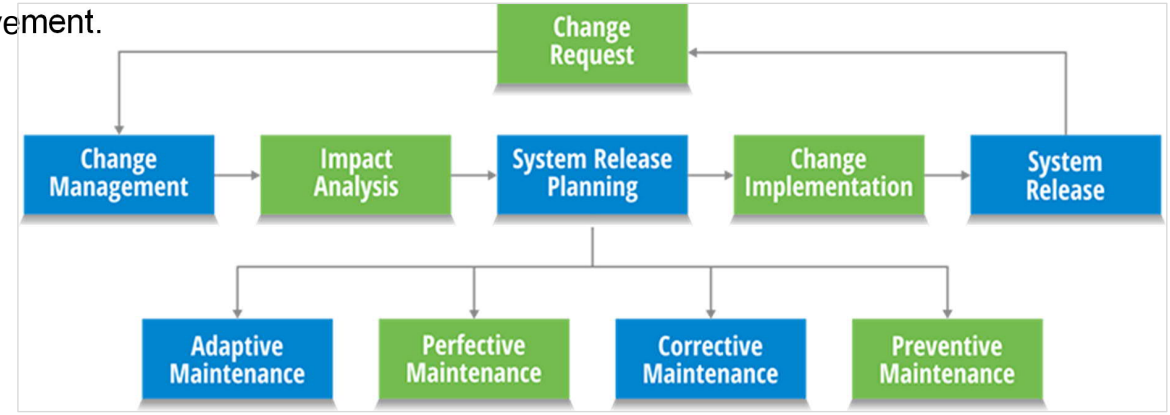
Note 1 to entry: Maintainability includes installation of updates and upgrades. Modifications include corrections, improvements or adaptation of the software to changes in environment, and in requirements and functional specifications. Modifications include those carried out by specialized support staff, and those carried out by business or operational staff, or end users.

Maintenance

- maintenance = make modifications to code for its extended life; add, change, delete.

Why maintenance

- bug fixing
- capability enhancement
- removal of outdated functrions
- performance improvement.



[https://simplified-it-outsourcing.com/blog/why-software-maintenance-is-necessary]

Maintenance

Sometimes it has been said, that in big projects developer company may sell the system in a cheap price, and they calculate they will get the profits later from maintenance.

Usually a tailor-made software system has ”continuous” or many years long contract about maintenance. Notice ”vendor lock”, manufacturer only is able to maintain.

Warranty is typically 6..12 months from deployment. During that time, developer company fixes those bugs which customer has found. Other modifications than bug fixes will be charged with normal prices.

Tampereen yliopisto
Tampere University

Why new (= bigger) software is slower than pervious versions... too many ”less necessary” new features, just to enhance selling ?

Tampereen yliopisto
Tampere University

QA = quality assurance

You can not add ”quality” to a product. Quality must be present from the very beginning in the process.

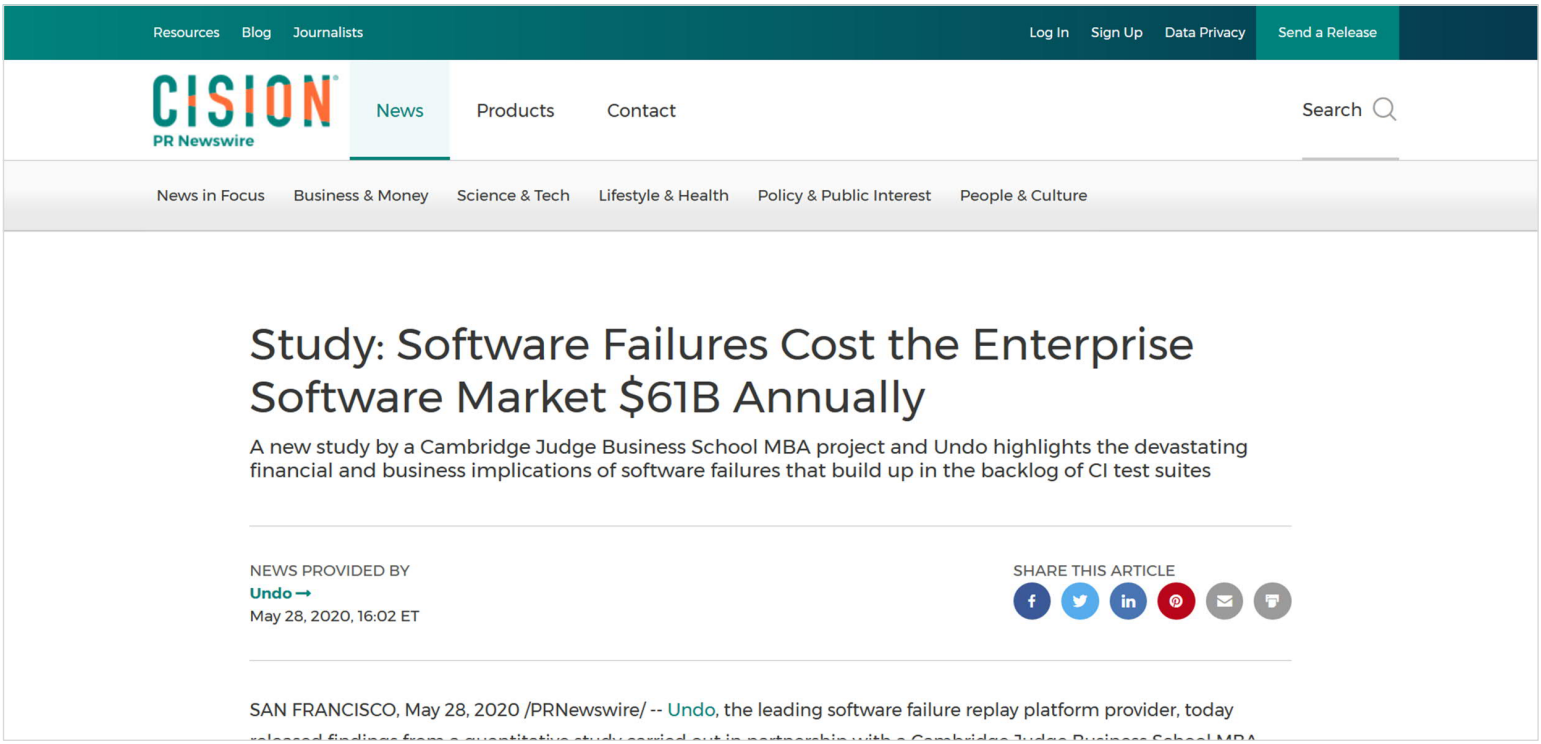
3.3259 , quality

- 1. degree to which the system satisfies the stated and implied needs of its various stakeholders, and thus provides value
- 2. ability of a product, service, system, component, or process to meet customer or user needs, expectations, or requirements
- 3. the degree to which a set of inherent characteristics fulfils requirements.

[ISO 24765:2017]

Tampereen yliopisto
Tampere University

[https://www.prnewswire.com/news-releases/]



What is ”quality software” ?

- easy for end-users to understand its use
- easy to maintain (e.g. add funtionality later)
- good code comments
- good architecture (e.g. easy to add components later)
- nice GUI (graphical user interface)
- end-user can do complex functions with a few mouse clicks
- 5000 users at a same time without delays (web application)
- cheap
- delivered in (or even ahead of) time
- it crashes or jams only once in a work-day ?

Well... difficult to describe.

It depends on the stakeholder from who you ask.

INTERNATIONAL STANDARD **ISO/IEC 25010**

First edition 2011-03-01

Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models

QA , QC

3.3260 , quality assurance (QA)

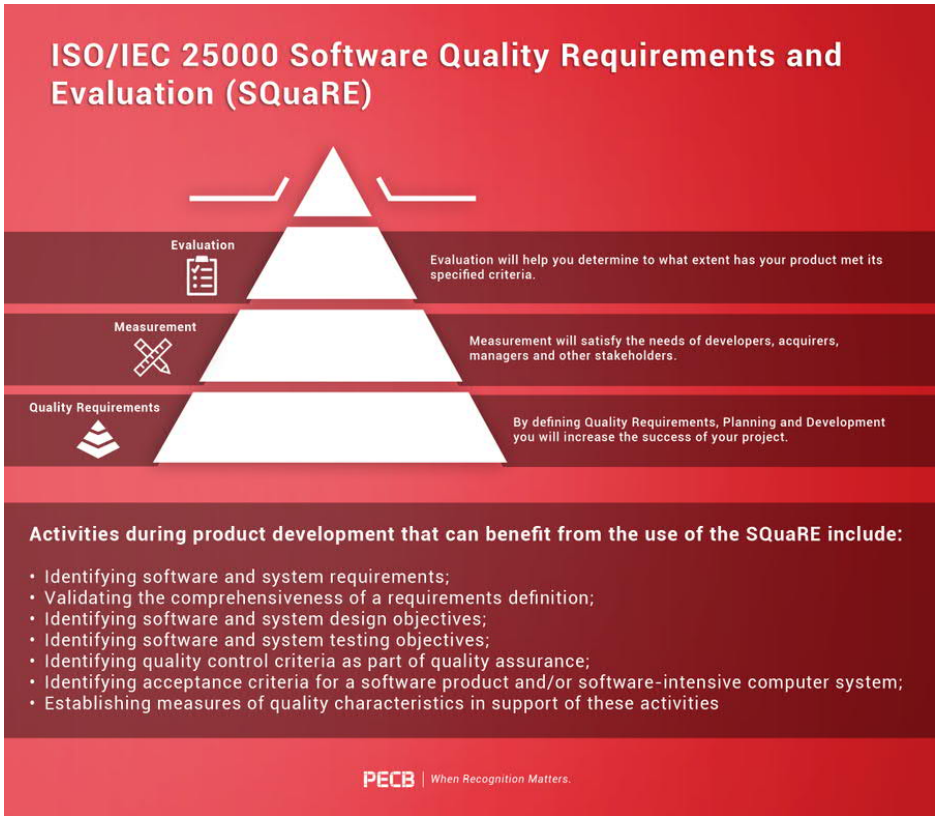
- 1. part of quality management focused on providing confidence that quality requirements will be fulfilled
- 2. all the planned and systematic activities implemented within the quality system, and demonstrated as needed, to provide adequate confidence that an entity will fulfill requirements for quality.

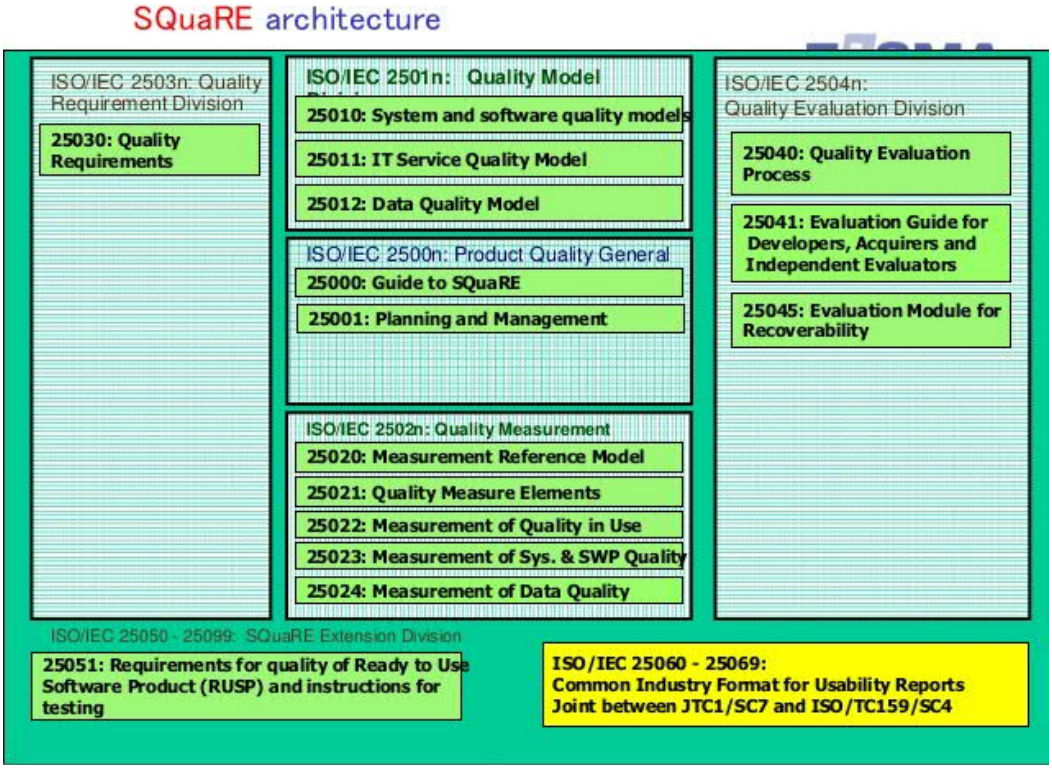
3.3265 , quality control (QC)

- 1. set of activities designed to evaluate the quality of developed or manufactured products
- 2. monitoring service performance or product quality, recording results, and recommending necessary changes.

[ISO 24765:2017]

Tampereen yliopisto
Tampere University





[https://www.slideshare.net/SuomenStandardisoimisliitto/isoiec-25000-square-measurement-of-product-quality-software-system-service-data]

There are many classifications of quality factors

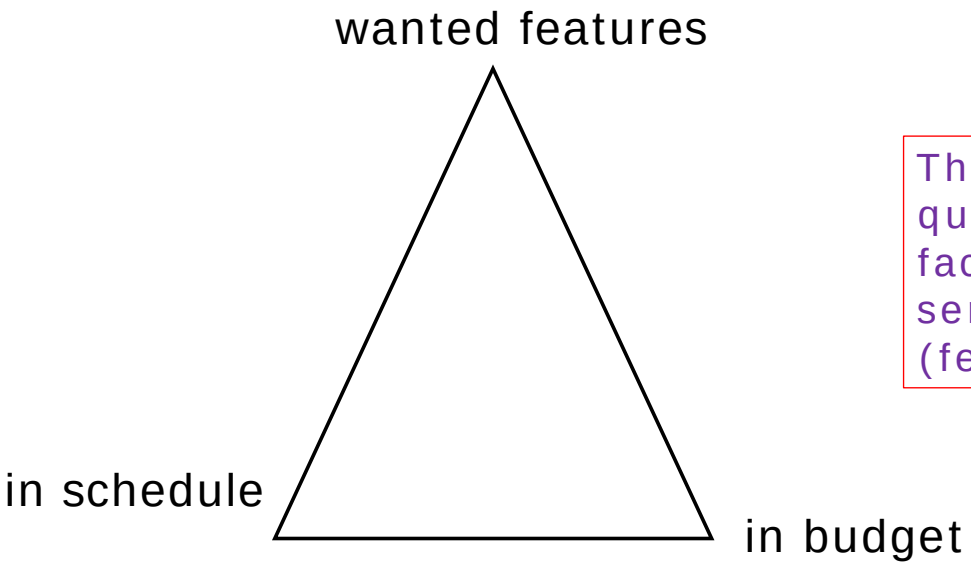
[Capgemini]

Software quality attributes [Sommerville, 2014]

Safety	Understandability	Portability
Security	Testability	Usability
Reliability	Adaptability	Reusability
Resilience	Modularity	Efficiency
Robustness	Complexity	Learnability

Quality is as customer and/or end-user feels it... do never forget them.

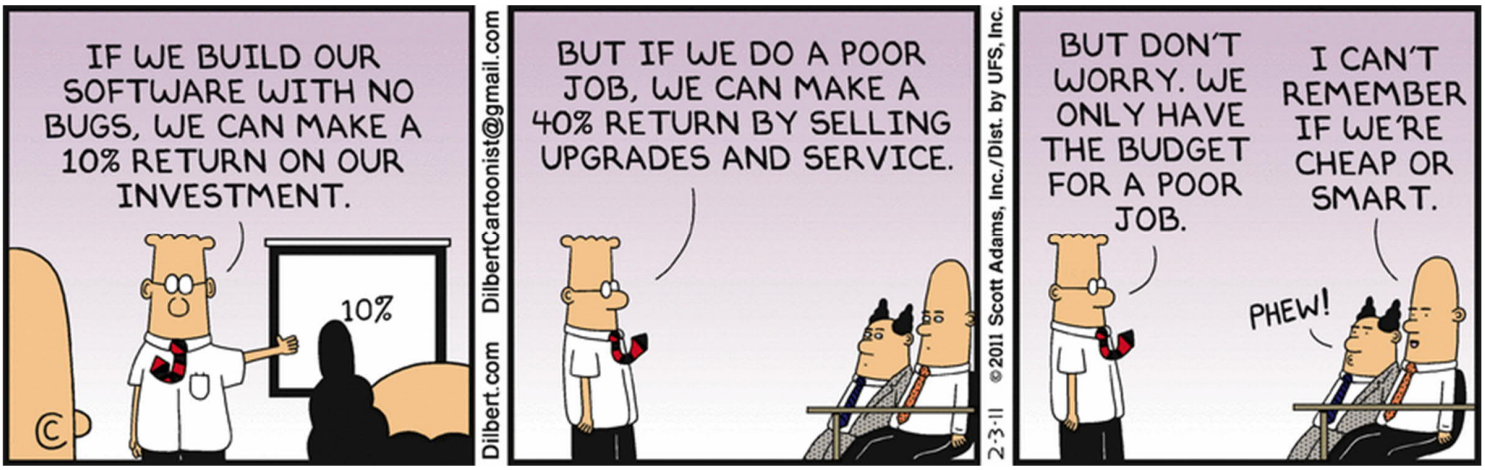
Remember what makes quality... four factors, customer gets sw project as



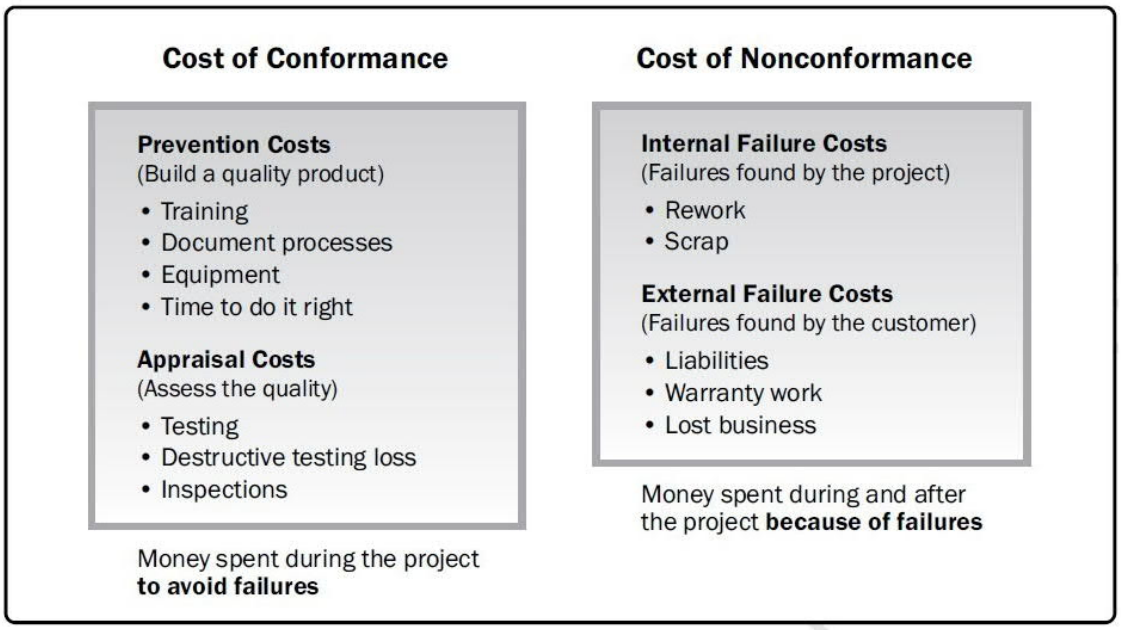
The 4th quality factor is: service ! (feeling)

Tampereen yliopisto
Tampere University

You could test forever, but at some moment you have to ship...

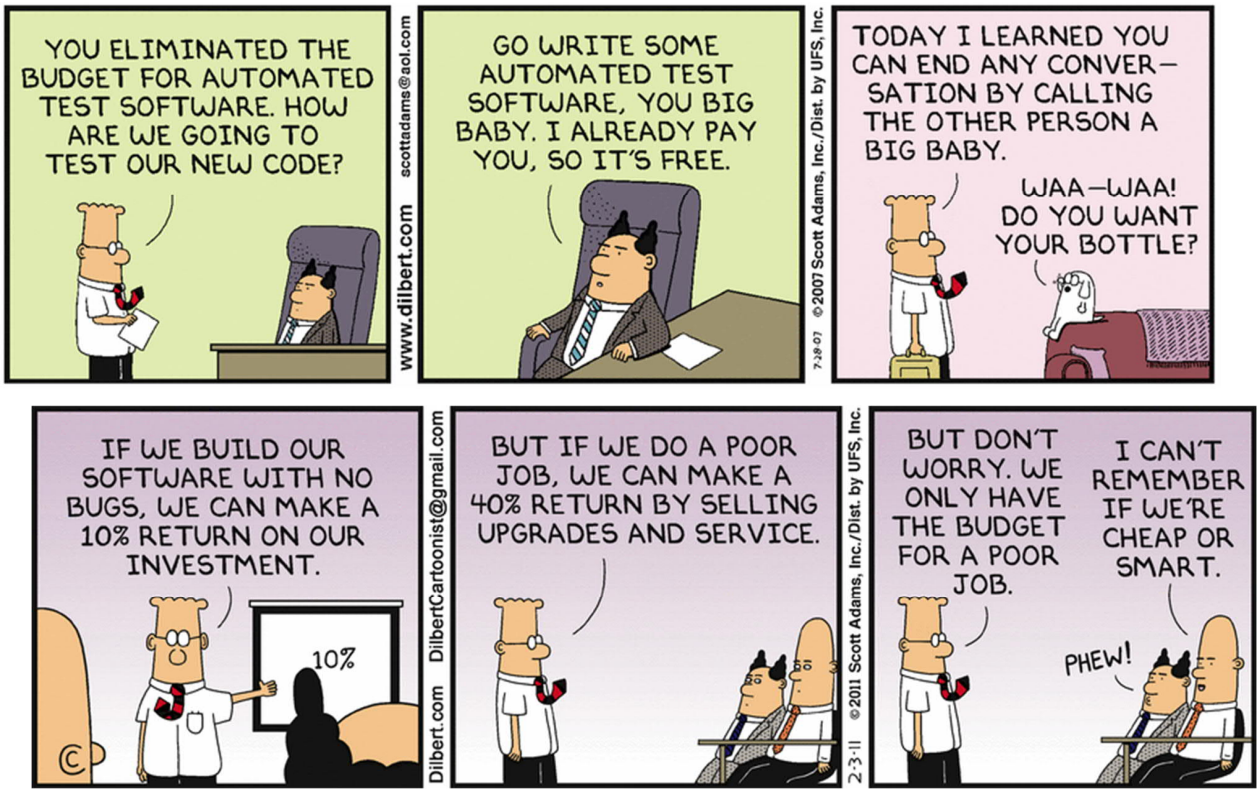


Quality costs in project



[Guide to PMBOK, 2017]

Figure 8-5. Cost of Quality



quality ISO 25000 standards set

Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE)

- ISO/IEC 25000 - Guide to SQuaRE
- ISO/IEC 25001 - Planning and Management
- ISO/IEC 25010 - System and software quality models
- ISO/IEC 25012 - Data Quality model
- ISO/IEC 25020 - Measurement reference model and guide
- ISO/IEC 25021 - Quality measure elements
- ISO/IEC 25022 - Measurement of quality in use
- ISO/IEC 25023 - Measurement of system and software product quality
- ISO/IEC 25024 - Measurement of data quality
- ISO/IEC 25030 - Quality requirements
- ISO/IEC 25040 - Evaluation reference model and guide
- ISO/IEC 25041 - Evaluation guide for developers, acquirers and independent evaluators
- ISO/IEC 25042 - Evaluation module
- ISO/IEC 25045 - Evaluation module for recoverability.

quality ISO 10000-standards set

- ISO 10001:2007, Quality management — Customer satisfaction — Guidelines for codes of conduct for organizations
- ISO 10002:2014, Quality management — Customer satisfaction — Guidelines for complaints handling in organizations
- ISO 10003:2007, Quality management — Customer satisfaction — Guidelines for dispute resolution external to organizations
- ISO 10004:2012, Quality management — Customer satisfaction — Guidelines for monitoring and measuring
- ISO 10005:2005, Quality management systems — Guidelines for quality plans
- ISO 10006:2003, Quality management systems — Guidelines for quality management in projects
- ISO 10007:2003, Quality management systems — Guidelines for configuration management
- ISO 10008, Quality management — Customer satisfaction — Guidelines for business-to-consumer electronic commerce transactions.

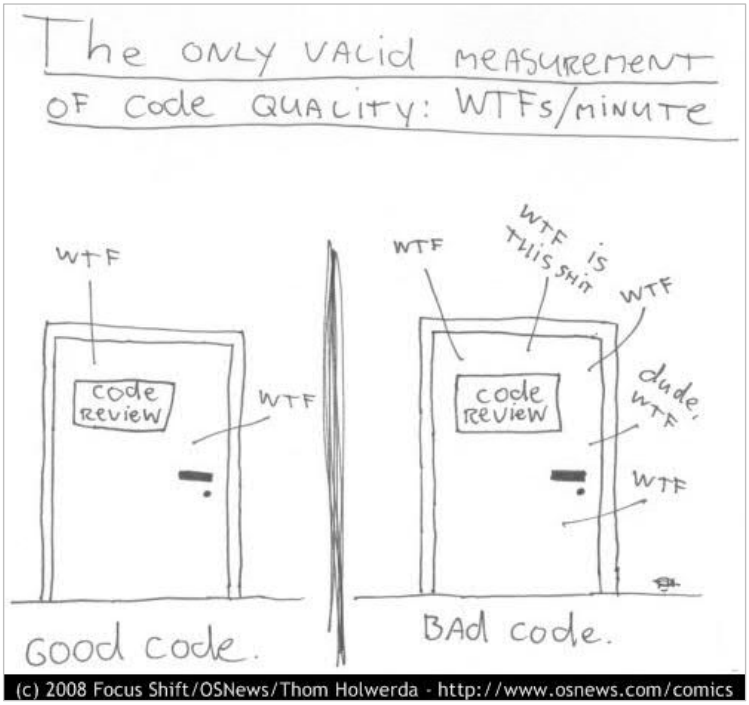
Inspections and reviews

Inspections and reviews

Inspection = formal, all material is scanned, log/minutes/diary is kept.

Review = informal, perhaps only main parts of deliverable is scanned, perhaps no any log just author's own memo.

Reviews and inspections for code should be done before Unit Testing. Although with automated unit tests you may test easily and inspect only the difficult or new parts of code.



3.2000 , inspection

1. visual examination of a software product to detect and identify software anomalies, including errors and deviations from standards and specifications

2. a static analysis technique that relies on visual examination of development products to detect errors, violations of development standards, and other problems3. examining or measuring to verify whether an activity,

component, product, result, or service conforms to specified requirements

cf. static testing

Note 1 to entry: Inspections are peer examinations led by impartial facilitators who are trained in inspection techniques. Determination of remedial or investigative action for an anomaly is a mandatory element of a software inspection, although the solution could be determined outside the inspection meeting. Types include code inspection and design inspection.

3.3503 , review

1. process, which can include a meeting, in which work products are presented to some stakeholders for comment or approval

2. process or meeting during which a software product is presented to project personnel, managers, users, customers, user representatives, or other interested parties for comment or approval

3. process or meeting during which a work product, or set of work products, is presented

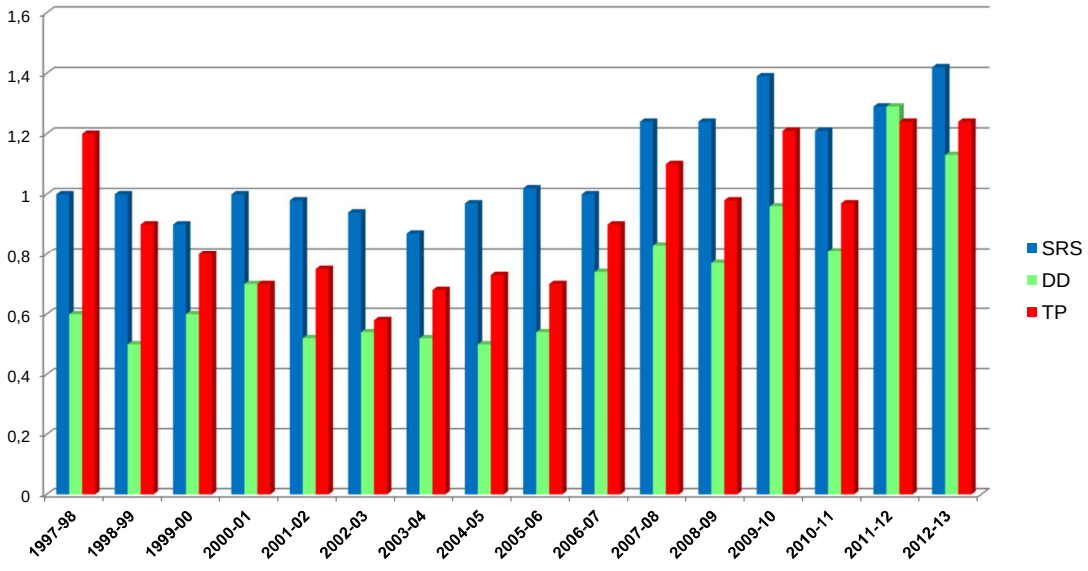
to project personnel, managers, users, customers, or other interested parties for comment or approval.

[ISO 24765-2017]

Why to test... what to test... surely you can not test everything

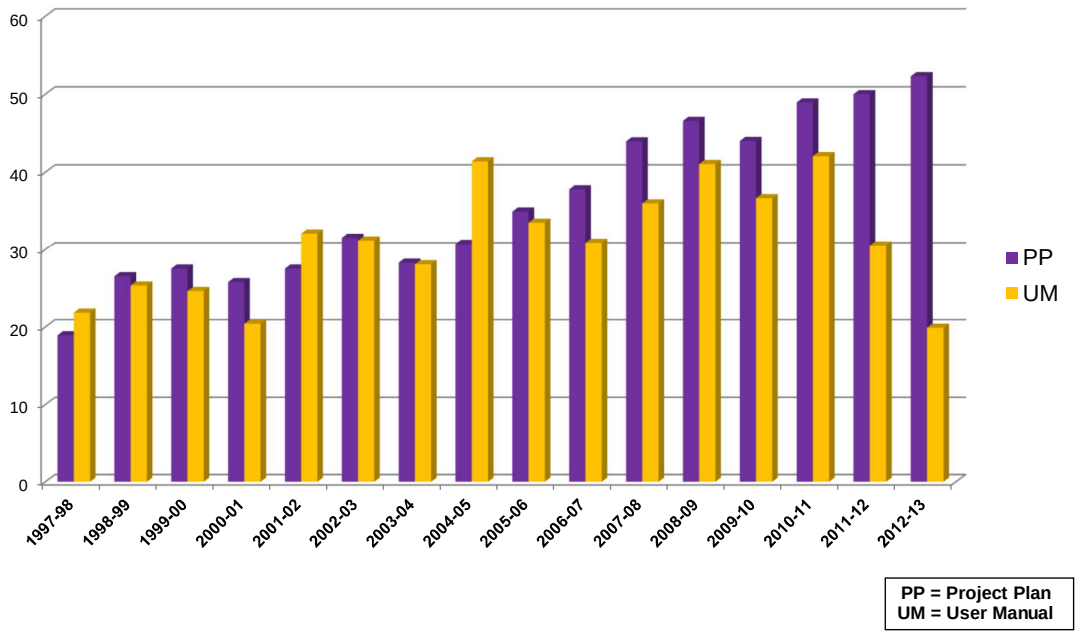
	A	B	C	D	E	F	G
1		INSPECTION DIARY (LIST OF FINDINGS)		Secretary: Teppo Teekkari			
2							
3		Group: 42. Hopeles:	Deliverable to be inspected: Project plan, v 1.1	Date: 20.10.2013			
4							
5		LOCATION (e.g. page/section/row)	EXPLANATION OF FINDING	SEVERITY	SOURCE	CORRECTED	VERIFIED
6	1	version history	versions' explanations are varying	K			
7	2	1.2	user groups missing	P			
8	3	1.3	add: inspection, review, status check	P			
9	4	1.4	is standard EN/SFS XYZ-131313:2011 really needed	S			
10							
11							
12							
13							
14							

Document inspections at TUT findings / page (average) (5/11)



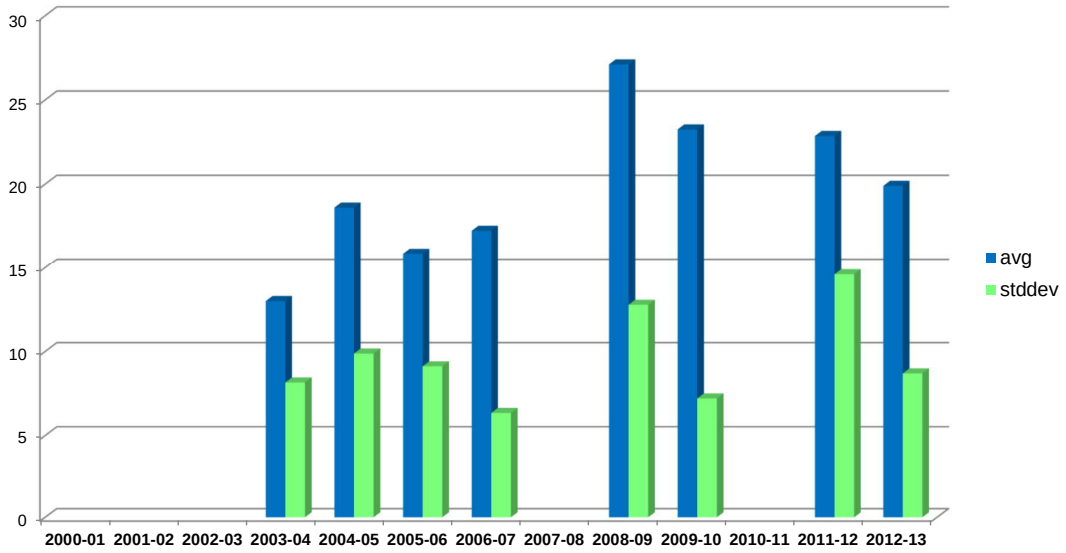
SRS= Sw Requirements Spec.
DD = Design Document
TP = Test Plan

Document inspections at TUT findings (average) (3/11)



PP = Project Plan
UM = User Manual

Code inspections at TUT findings (average) (7/11)



Motivation for inspections

- ~30 times more effective in defect finding than testing
- Early defect detection -> “right the first time”
- Spreads information to several people -> reduces risks in losing a key employee
- Effective in training new members in a group
- Results in a better common understanding of the product
- Cases report improved quality in defects, deliveries and schedules
- Manageability and productivity tend to improve as well as employee satisfaction
- Better commitment from the team to the project
- The most easiest and effective way of QA
- Inspections DO affect to product quality
  (but they should be done well) !

Tampereen yliopisto
Tampere University

Peer review, some colleague checks your work

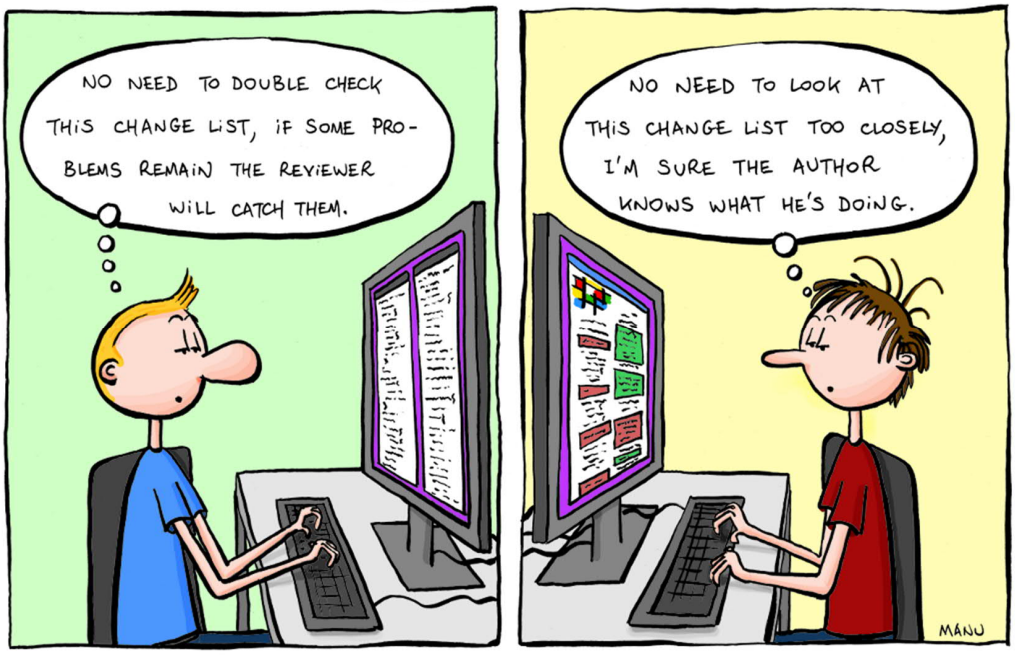
Using

Merge request (GitLab) or

Pull request (GitHub)

means that some developer actually checks (review) the code before adding it to e.g. master branch.

It is a kind of code collaboration.



THIS IS NOT THE WAY IT SHOULD GO !

Tampereen yliopisto
Tampere University

COMP.SE.100-EN (ItSE)
Introduction to Software Engneering

Lecture 8, 28.10.2020

Tensu: remember to pause Zoom lecture recording

Zoom lecture break, 10 minutes stretching, walking, etc.

Testing

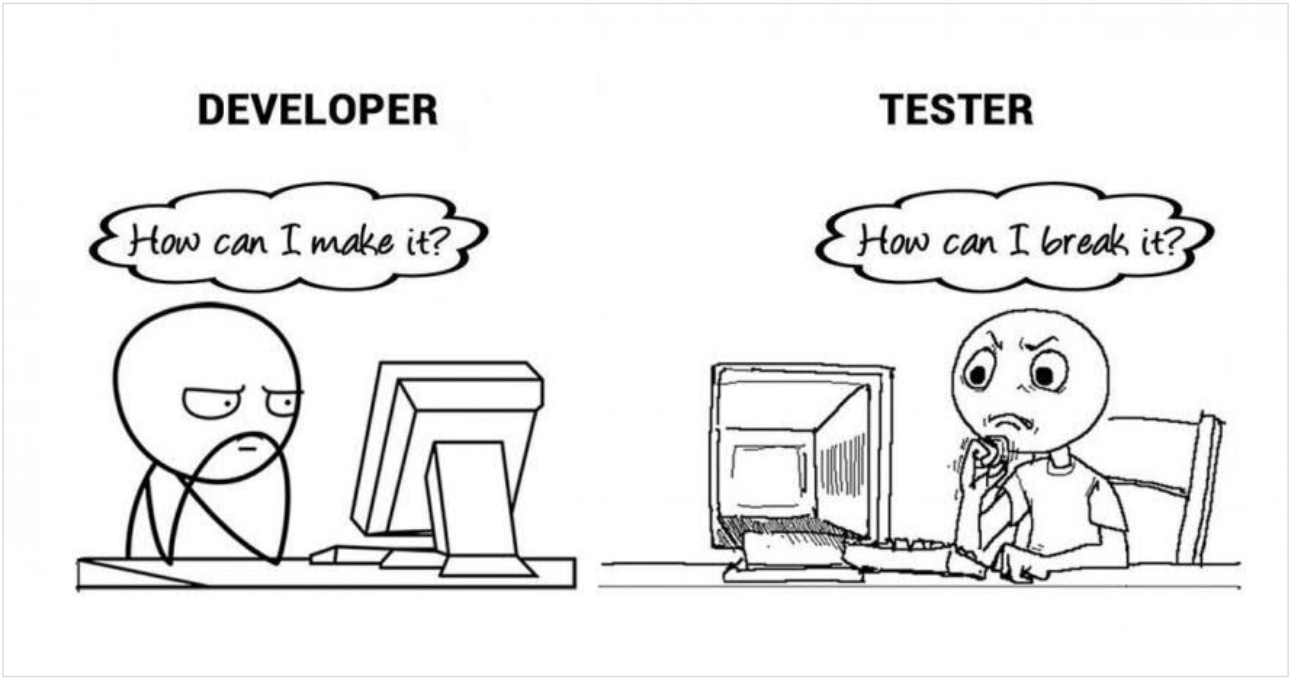
Well, some folklore

- if testing finds many bugs, code is poor
- if testing does not find many bugs, testing is poor.

So you (project manager) are doomed any way... ?

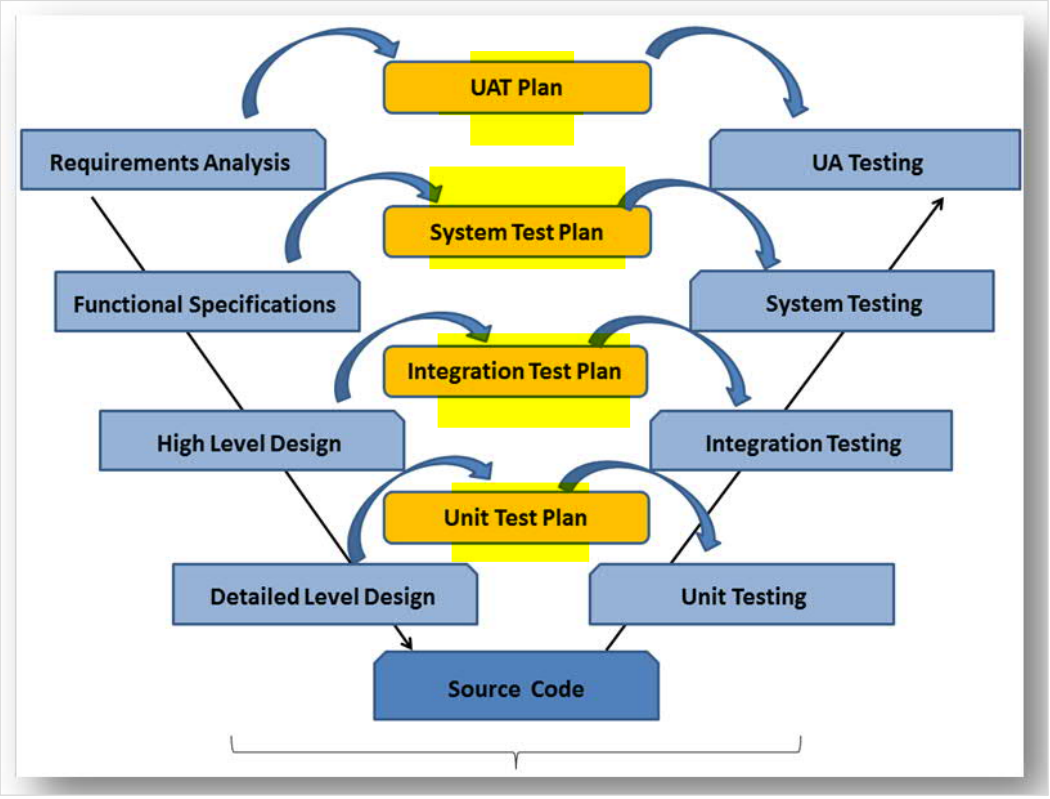
What about from the point of view of a customer ?

Remember that ALL software has errors... but hopefully not many, and not fatal/critical/major/blocking.



Testing phases

V-model



[qainsights.com/]

Tampereen yliopisto
Tampere University

Folklore: in Finland one company had a short time bonus for developers, as how many lines of code (LOC) they write. Well, you know human nature… it did not last much over one month.

https://www.tutorialspoint.com/difference-between-testing-and-debugging

Following are the important differences between Testing and Debugging.

Sr. No.	Key	Testing	Debugging
1	Definition	Technically Testing is a process to check if the application is working same as it was supposed to do, and not working as it was not supposed to do.	On other hand Debugging is the activity performed by developers to fix the bug found in the system.
2	Objective	Main objective of Testing is to find bugs and errors in an application which get missed during the unit testing by the developer.	On other hand the main objective of Debugging is to find the exact root cause at code level to fix the errors and bugs found during the testing.
3	Perform	As Testing is mainly to find out the errors and bugs is mainly performed by the testers. Also if testing is at developer end known as unit testing then it is performed by the Developer.	While on other hand Debugging is to find the missing or de-faulty code in an application hence major performed by the developers only.

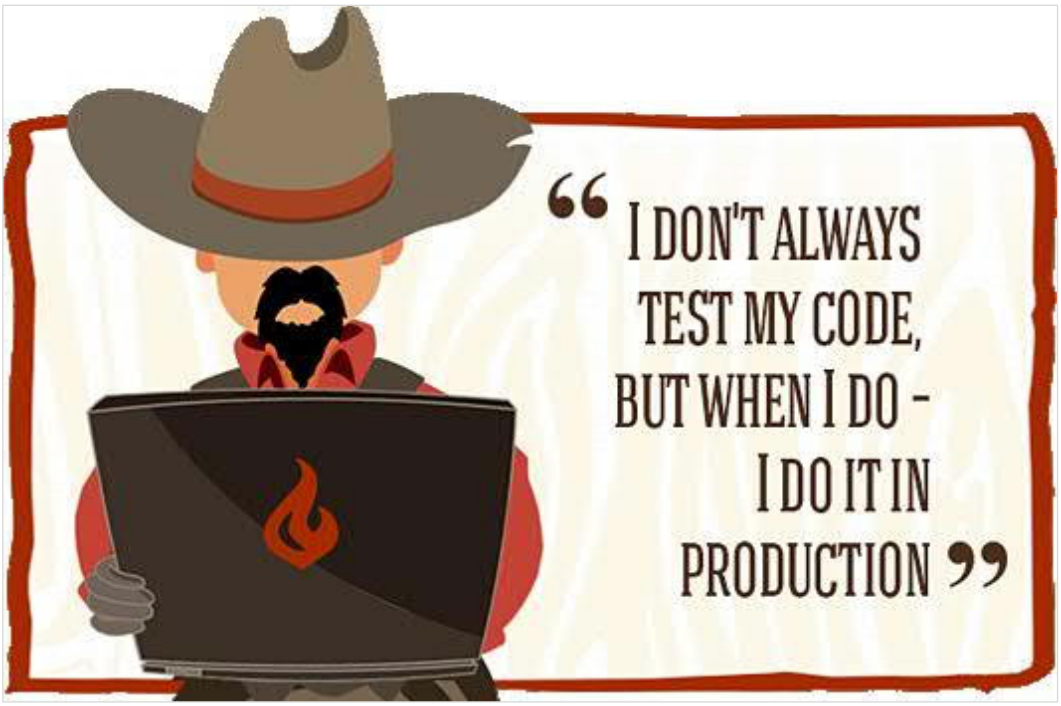
Usual kinds of testing

Remember also reviews and inspections, before testing.

- unit testing, earlier module testing (one component)
- smoke testing (is it stable enough for system testing)
- system testing (the whole product)
- integration testing, several modules/components are tested together
- reggression testing (run old tests again, and compare new to old results)
- usability testing (by actual end-users, or specialists test), e.g. exploratory testing (FI: tutkiva testaus)
- special testing; security testing, stress testing (e.g. server load),...
- UAT = user acceptance testing
- acceptence testing (the final delivered system).

Use some issue tracker or bug database to keep track on errors and corrections.

”Cowboy coder”



Testing distinction (one classification)

Please note that Software Testing Types are different from Levels or Methods.

In contrast to the Software Testing Types as different viewpoints, Software Testing Levels are the tests done at various stages of software development and
Software Testing Methods are the ways the tests are conducted.

For example, you can do Functional Testing (A Type) during System Testing (A Level) using Black Box Testing (A Method).

[softwaretestingfundamentals.com/software-testing-types/]

Software testing types

Smoke Testing Smoke Testing, also known as “Build Verification Testing”, is a type of software testing that comprises of a non-exhaustive set of tests that aim at ensuring that the most important functions work.

Functional Testing Functional Testing is a type of software testing whereby the system is tested against the functional requirements/specifications.

Usability Testing Usability Testing is a type of software testing done from an end-user’s perspective to determine if the system is easily usable.

Security Testing Security Testing is a type of software testing that intends to uncover vulnerabilities of the system and determine that its data and resources are protected from possible intruders.

Performance Testing Performance Testing is a type of software testing that intends to determine how a system performs in terms of responsiveness and stability under a certain load.

Regression Testing Regression testing is a type of software testing that intends to ensure that changes (enhancements or defect fixes) to the software have not adversely affected it.

Compliance Testing Compliance Testing [also known as conformance testing, regulation testing, standards testing] is a type of testing to determine the compliance of a system with internal or external standards.

[softwaretestingfundamentals.com/software-testing-types/]

Software testing levels

Unit Testing A level of the software testing process where individual units of a software are tested. The purpose is to validate that each unit of the software performs as designed.

Integration Testing A level of the software testing process where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units.

System Testing A level of the software testing process where a complete, integrated system is tested. The purpose of this test is to evaluate the system’s compliance with the specified requirements.

Acceptance Testing A level of the software testing process where a system is tested for acceptability. The purpose of this test is to evaluate the system’s compliance with the business requirements and assess whether it is acceptable for delivery.

Note: Some tend to include Regression Testing as a separate level of software testing but that is a misconception. Regression Testing is, in fact, just a type of testing that can be performed at any of the four main levels.

[softwaretestingfundamentals.com/software-testing-levels/]

Software testing methods

Black Box Testing A software testing method in which the internal structure/design/implementation of the item being tested is not known to the tester. These tests can be functional or non-functional, though usually functional. Test design techniques include Equivalence partitioning, Boundary Value Analysis, Cause-Effect Graphing.

White Box Testing A software testing method in which the internal structure/design/implementation of the item being tested is known to the tester. Test design techniques include Control flow testing, Data flow testing, Branch testing, Path testing. (Also known as glass box testing.)

Gray Box Testing A software testing method which is a combination of Black Box Testing method and White Box Testing method.

Agile Testing A method of software testing that follows the principles of agile software development.

Ad Hoc Testing A method of software testing without any planning and documentation.

[softwaretestingfundamentals.com/software-testing-methods]

Tampereen yliopisto
Tampere University

Testing artifacts

Software testing usually produces testing artifacts mentioned below:

- Test Data – multiple sets of values to be used as inputs for testing definite functionality often combined into one file.
- Test Script – code that substitutes user activity and/or interaction with software UI.
- Test Case – consists of preconditions, steps, inputs, and expected results to test some part of the functionality.
- Test Scenario – test case with higher level of abstraction that depicts scenarios in which user is considered to use the software.
- Test Suite – the set of test cases or test scenarios for given functionality or testing type (i.e. regression, smoke, or sanity).
- Test Plan – document that depicts testing tactics to test definite software product in the definite testing run; often consists of the test suites to be executed and the testing approach to be used.

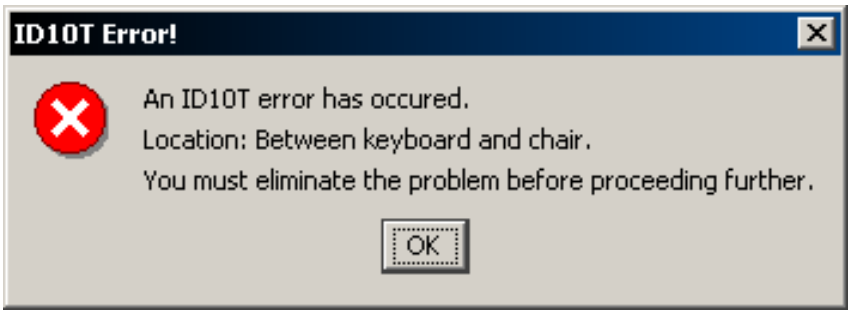
[Kulesovs et al., Inventory of Testing Ideas and Structuring of Testing Terms, 2013]

Tampereen yliopisto
Tampere University

Do NOT take all testing (or comics) literally.

Tampereen yliopisto
Tampere University

ID-10-T error (FI: ID sata TTI -virhe)

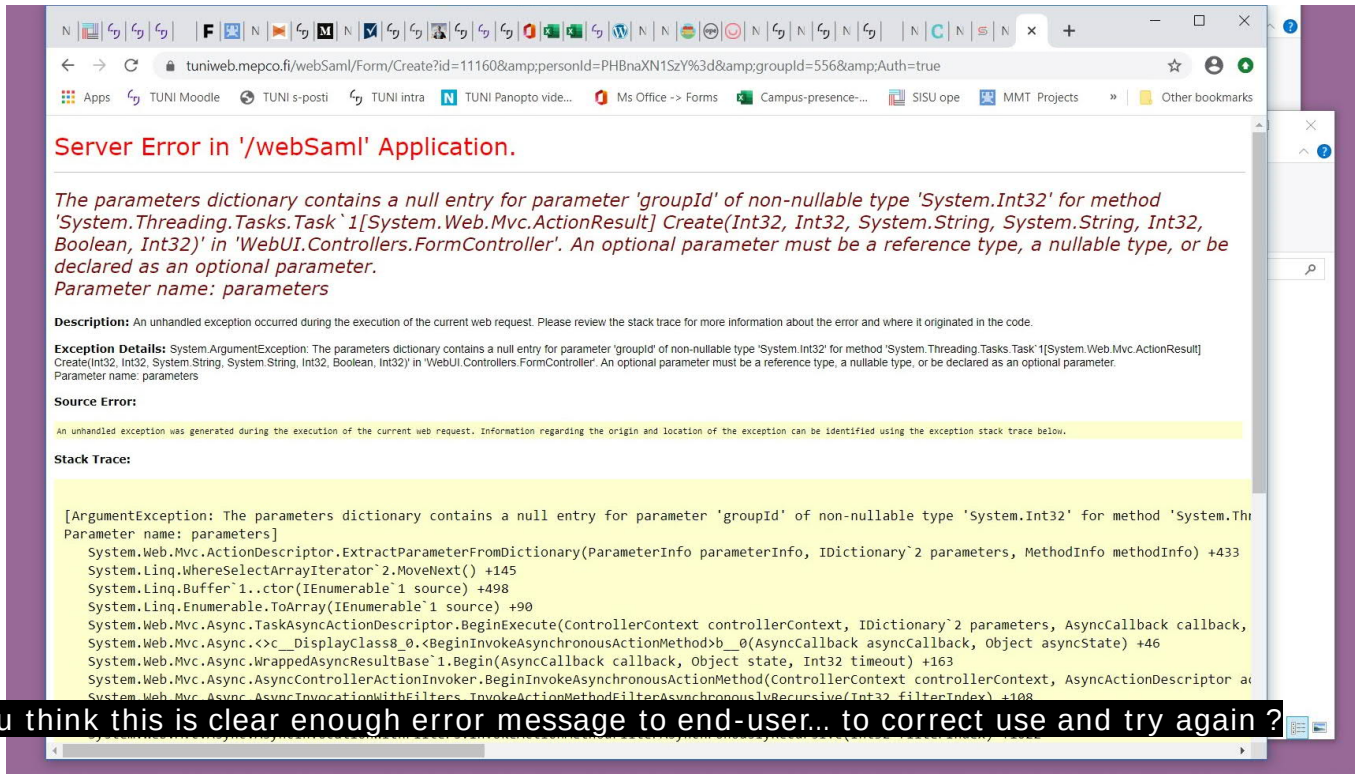


This is NOT a test plan: "We will do some system testing, if we have time."

Sw eng folklore...

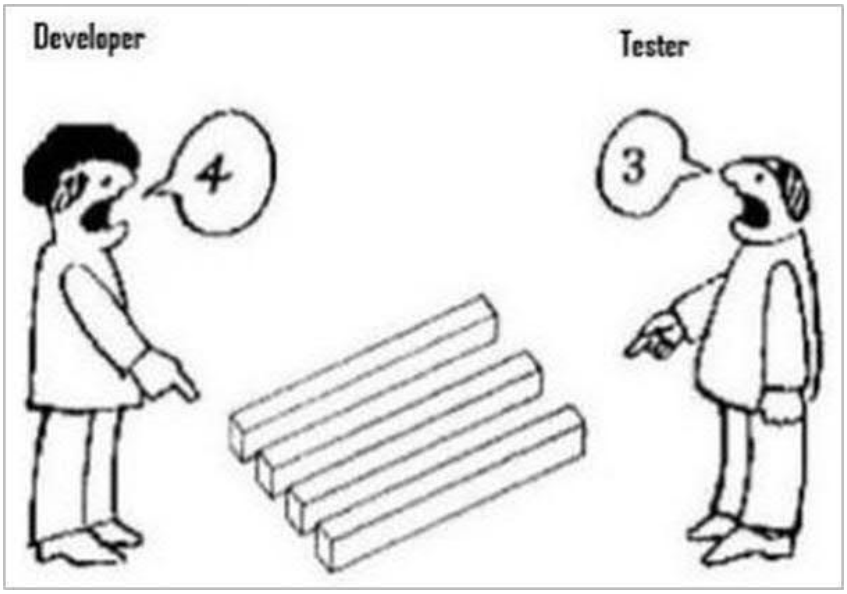
Project size (number of lines of code)	Average error density
less than 2K	0 - 25 errors per 1000 lines of code
2K - 16K	0 - 40 errors per 1000 lines of code
16K - 64K	0.5 - 50 errors per 1000 lines of code
64K - 512K	2 - 70 errors per 1000 lines of code
512K and more	4 - 100 errors per 1000 lines of code

Tampereen yliopisto
Tampere University

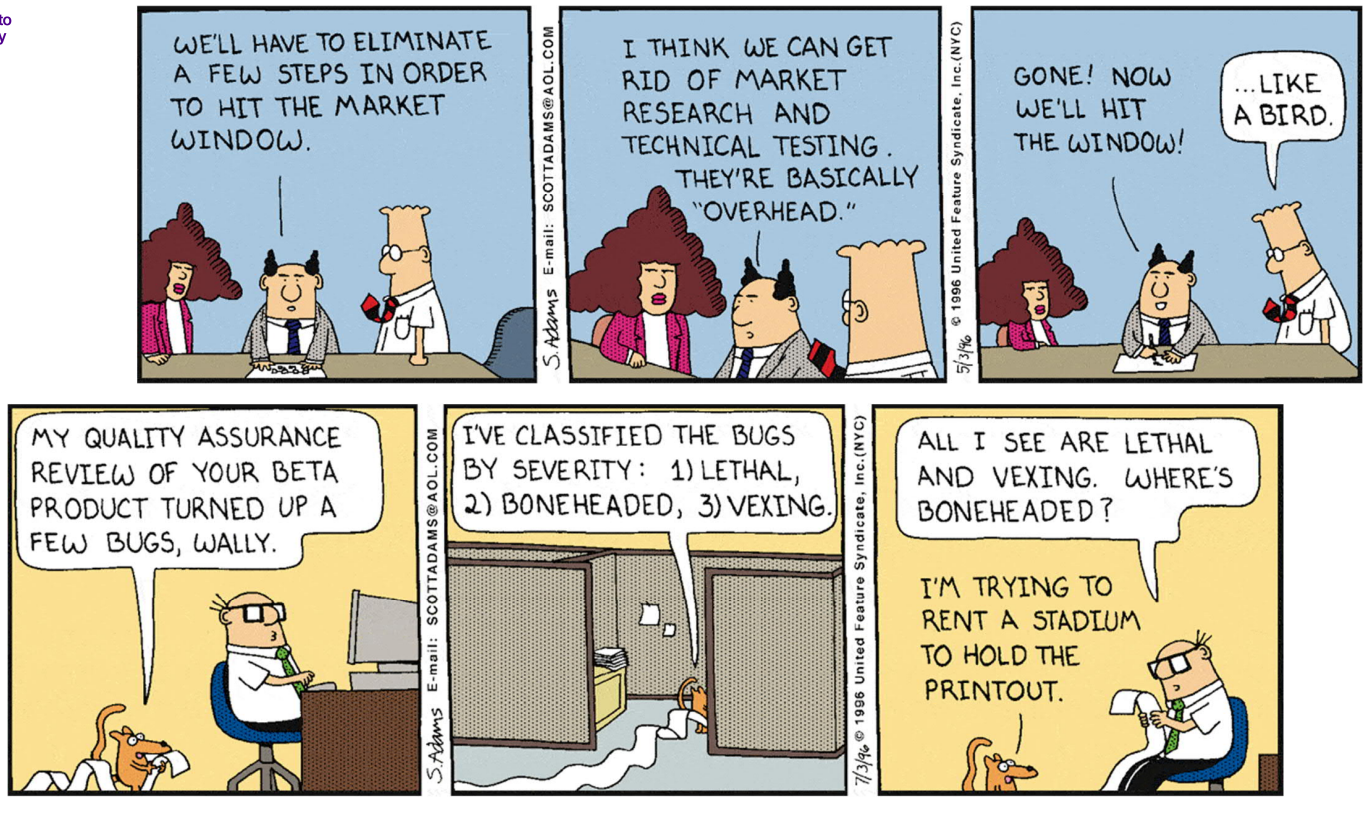


Do you think this is clear enough error message to end-user... to correct use and try again ?

Who is right ?
The one who pays for the work ?



Tampereen yliopisto
Tampere University



Error.. or what... terminology varies

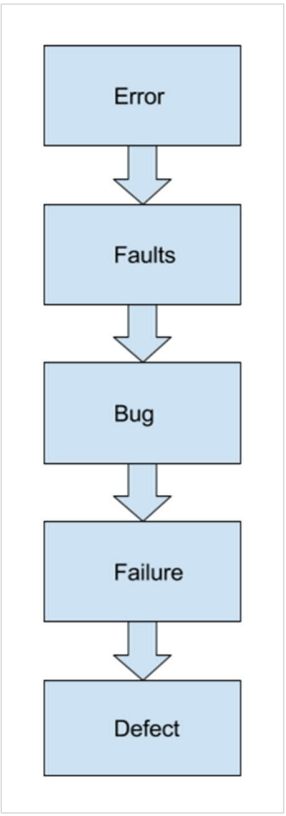
Error is a deviation from the actual and the expected result. It represents the mistakes made by the people.

Faults are the result of an error. It is the incorrect step or process due to which the program or the software behaves in an unintended manner

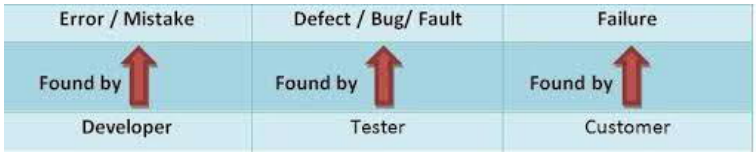
Bug is an evidence of Fault in a program due to which program does not behave in an intended manner

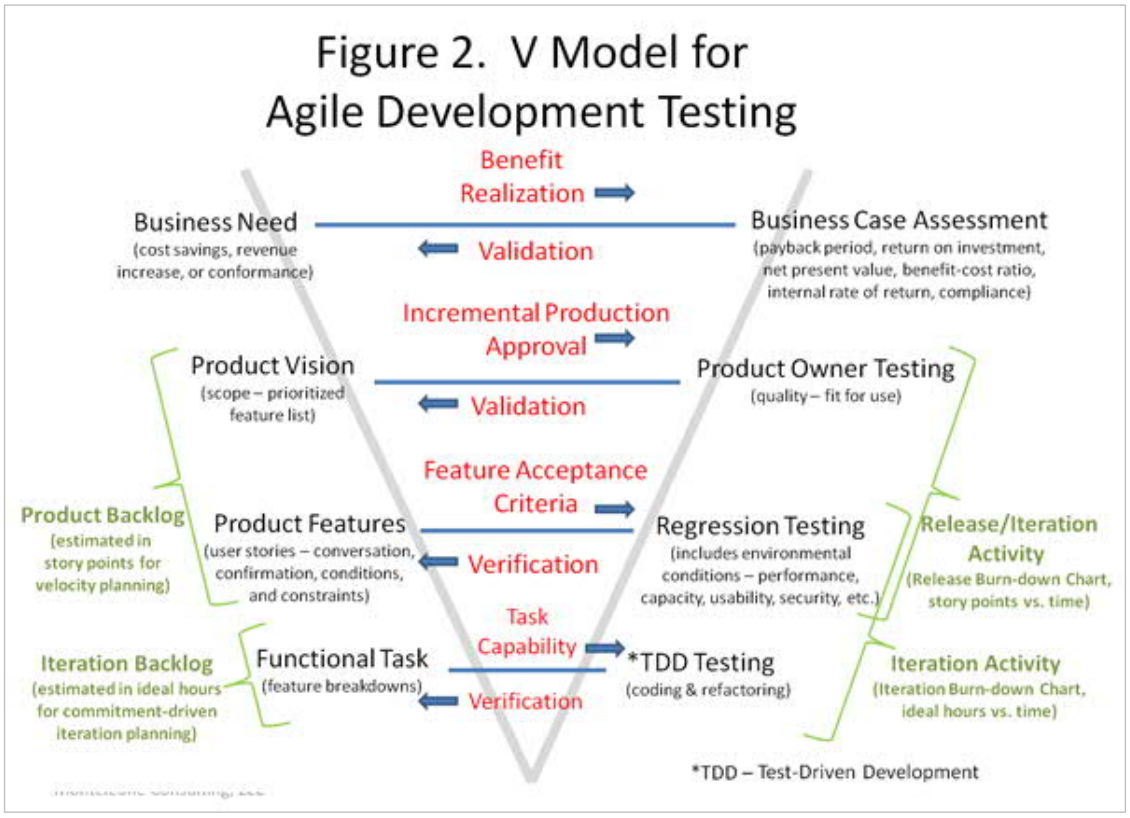
Failure is an inability of the system or components to perform its required function. Failure occurs when Faults executes

Defect is said to be detected when Failure occurs.

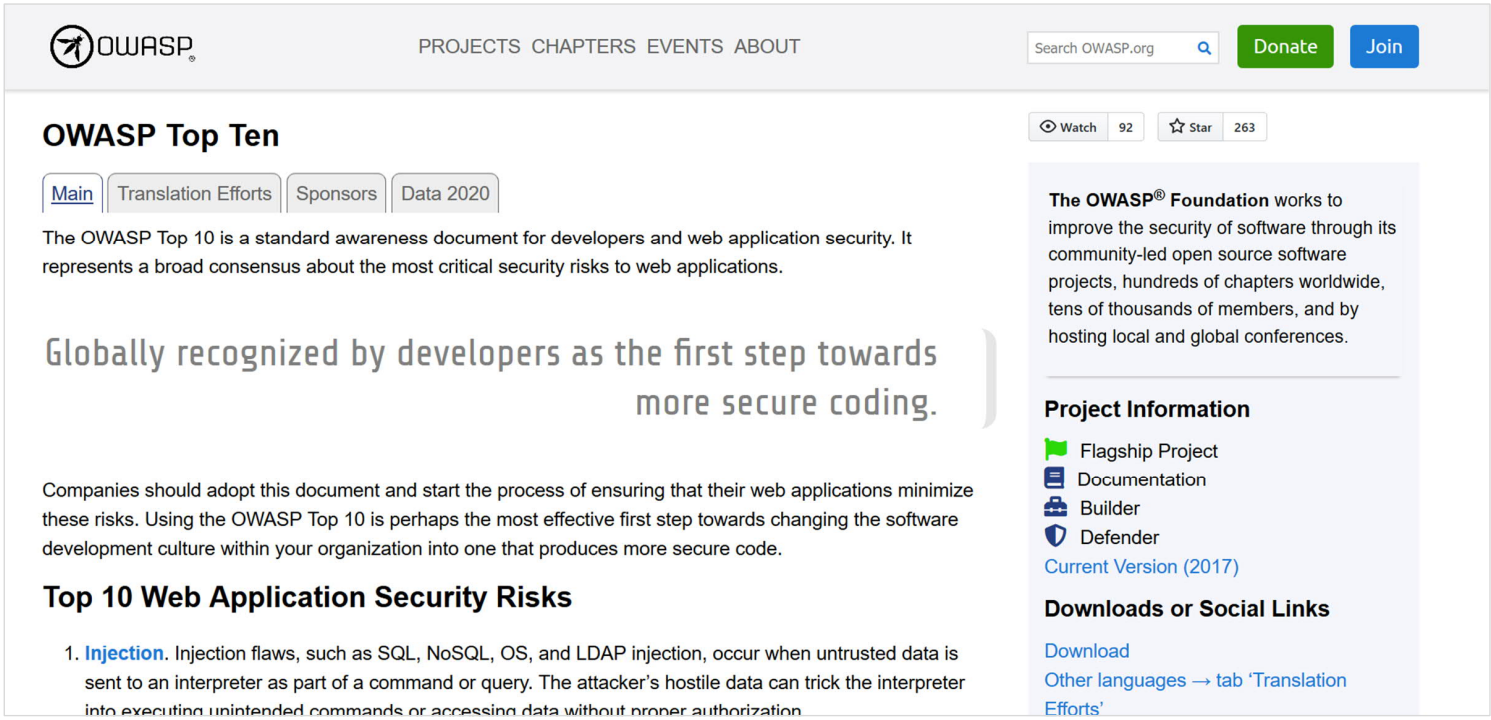


[stackoverflow.com/]

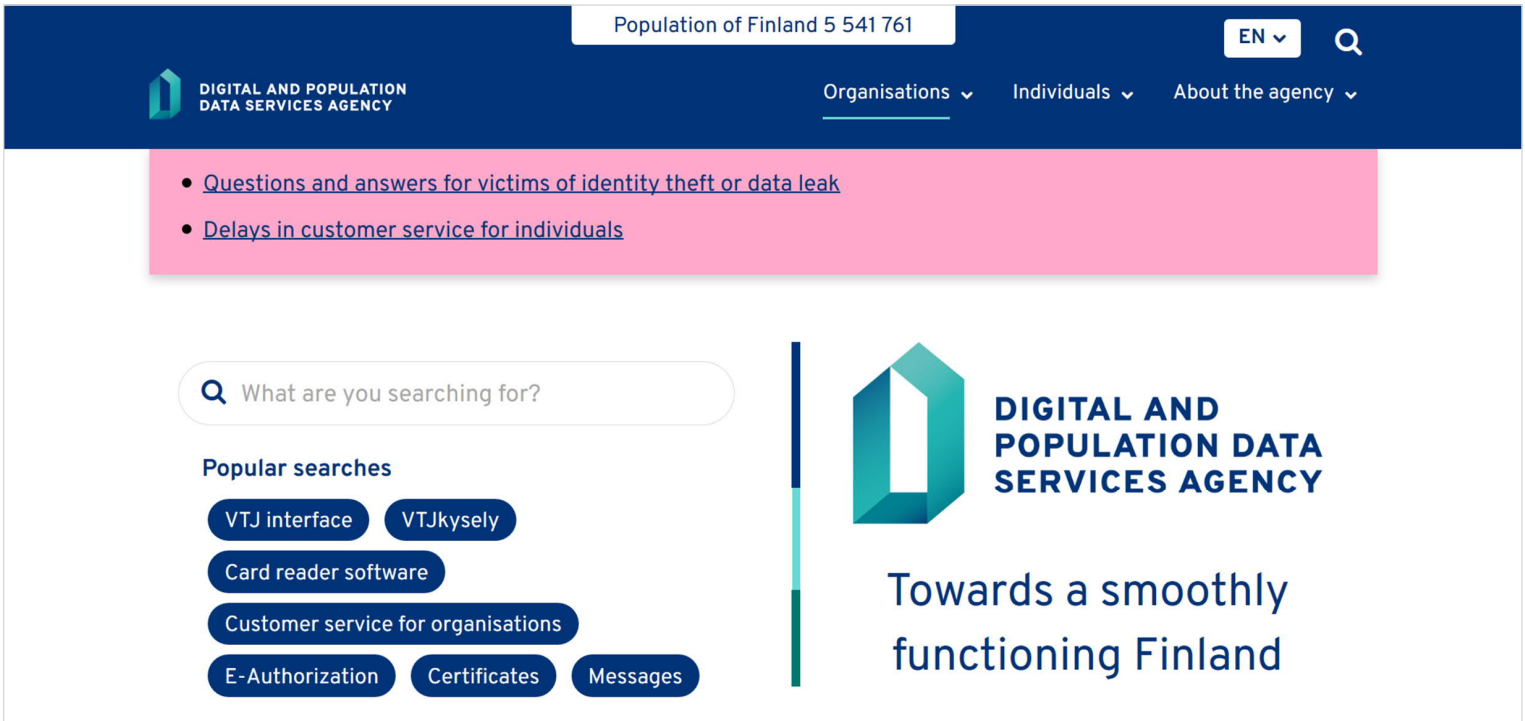




OWASP = The Open Web Application Security Project



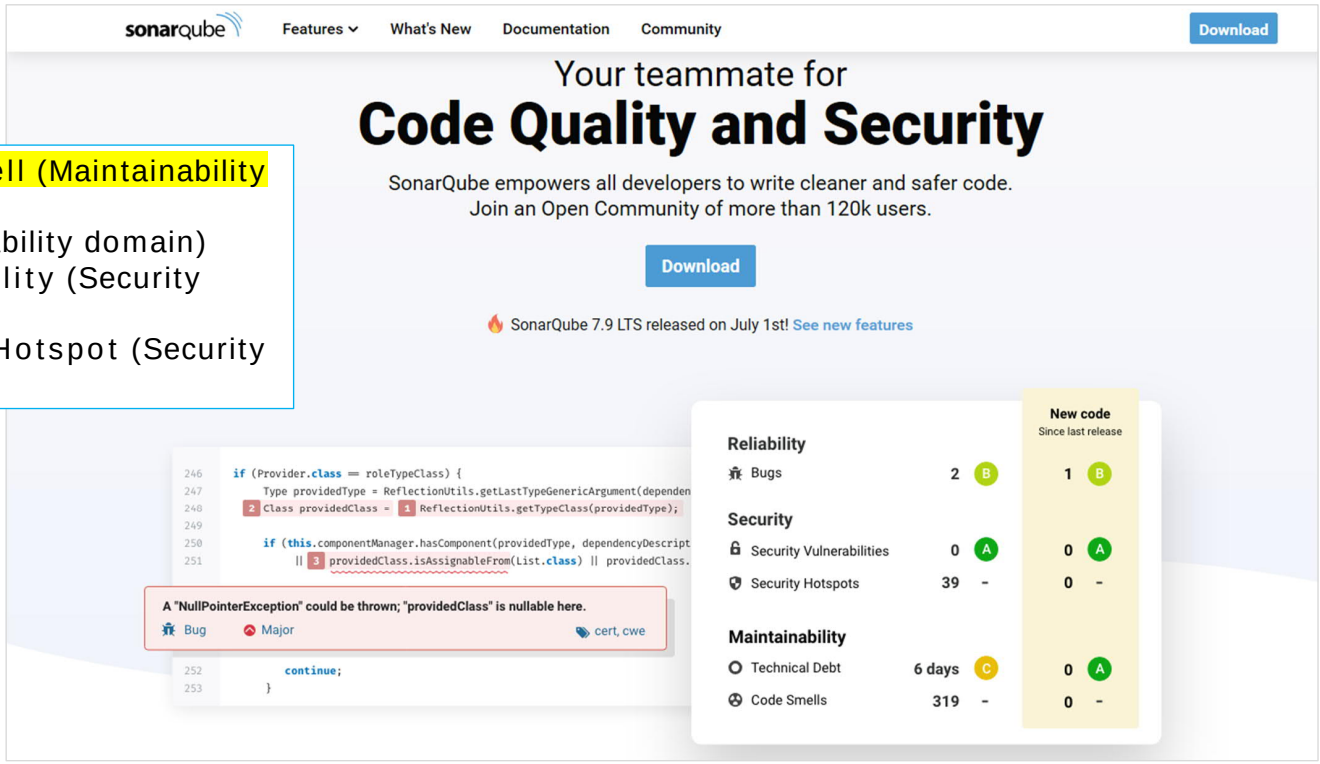
[https://dvv.fi/vahti]

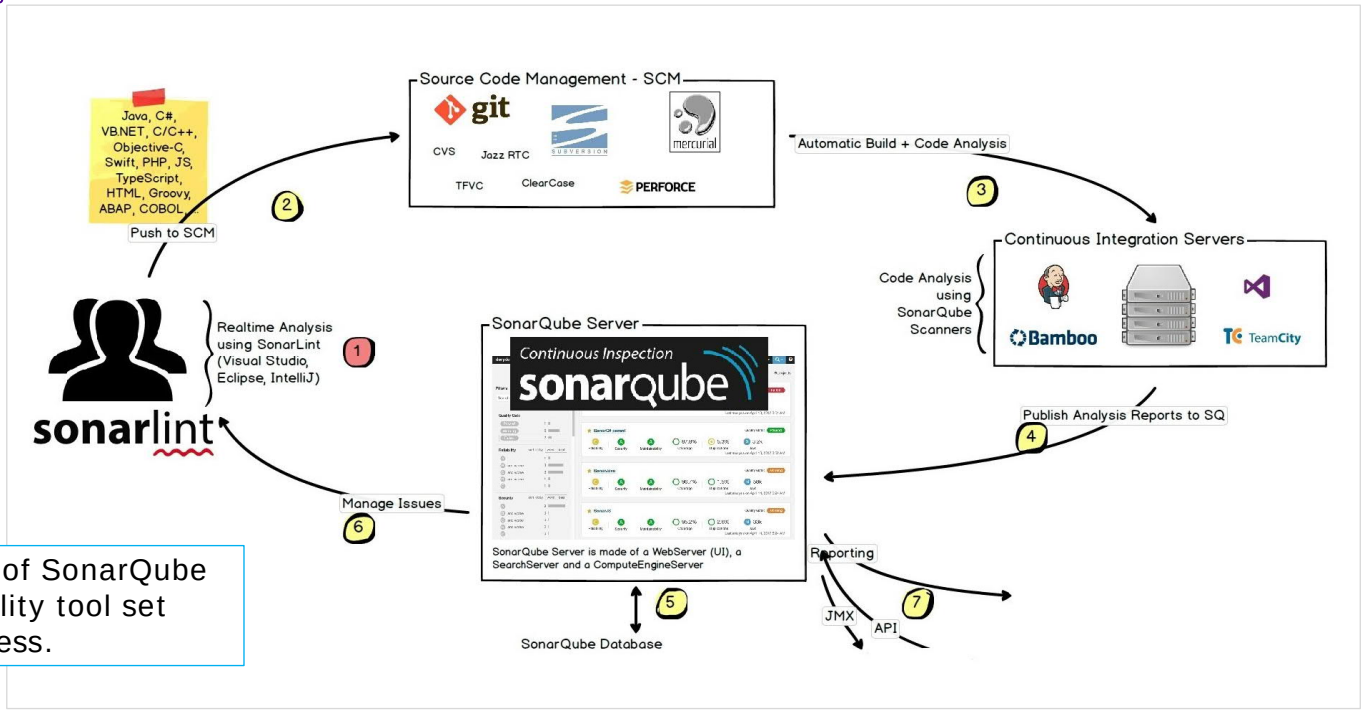


Tampereen yliopisto
Tampere University

Static code analysing ”code smells”

- Code Smell (Maintainability domain)
- Bug (Reliability domain)
- Vulnerability (Security domain)
- Security Hotspot (Security domain)





Example of SonarQube code quality tool set and process.

SonarQube tool, some measurements

Project, e.g. ”Sonar Groovy” or ”sonar-test” (in GitHub)

- Home (bugs&vulnerabilities, code smells, coverage, duplications)
- Issues
  - Type (bug, vulnerability, code smell)
  - Resolution (unresolved, fixed, false positive, won’t fix, removed)
- Measures
  - Reliability (bugs.. rating … effort)
  - Security (vulnerabilities … rating … effort)
  - Maintainability (code smells … rating … technical debt)
  - Coverage (line … condition … uncovered … unit tests … duration)
  - Duplications (blocks … lines … files)
  - Size (LOC … lines … statemenmts … functions … classes … files … directories … comments)
  - Complexity (function … file … class)
  - Issues
- Code (common numbers shortly)
- Activity (by date).

https://blog.codinghorror.com/code-smells/

- Developing your "code nose" is something that happens early in your programming career, if it's going to happen at all.
- Combined code; most of these smells should be familiar to you.

Code Smells Within Classes

- Comments
- Long Method
- Long Parameter List
- Duplicated code
- Conditional Complexity
- Combinitorial Explosion
- Large Class
- Type Embedded in Name
- Uncommunicative Name (method)
- Inconsistent Names
- Dead Code
- Speculative Generality
- **Oddball Solution**
- **Temporary Field.**

Mika Mäntylä: code smell

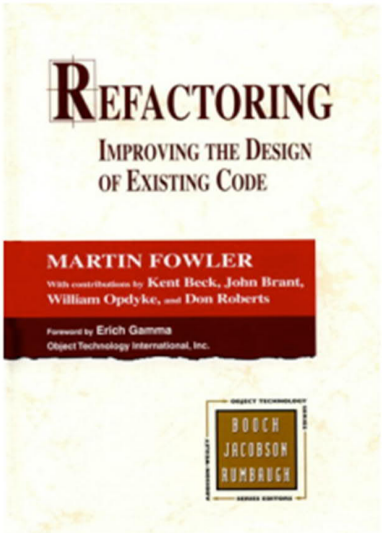
Group name (Smells in group)

- The Bloaters (Long Method, Large Class, Primitive Obsession, Long Parameter List, DataClumps)
- The Object-Orientation Abusers (Switch Statements, Temporary Field, Refused Bequest, Alternative Classes with Different Interfaces)
- The Change Preventers (Divergent Change, Shotgun Surgery, Parallel Inheritance Hierarchies)
- The Dispensables (Lazy class, Data class, Duplicate Code, Dead Code, Speculative Generality)
- The Couplers (Feature Envy, Inappropriate Intimacy, Message Chains, Middle Man).

refactoring

Bad Code Smells and Refactoring

- Code smells
  - Indicative of bad software design
  - List of bad smells: http://blog.codinghorror.com/code-smells/
  - Useful "catalog" of refactorings: http://www.refactoring.com/catalog/
  - Mapping of smells to refactorings: http://www.industriallogic.com/wp-content/uploads/2005/09/smellstorefactorings.pdf



Tampereen yliopisto
Tampere University

Code style guide
Coding conventions

(may be extracted from quality handbook)

Tampereen yliopisto
Tampere University

Why to bother for code style

Well, customer doesn't see code style… unless some "funny" error messages pop up.

Maintainers curse missing comments, or chancing coding styles.

Ideally code in a large software would be like it was written by the one same person.

That way it is easy to understand and maintain.

Well, you have learned all that already in programming courses.

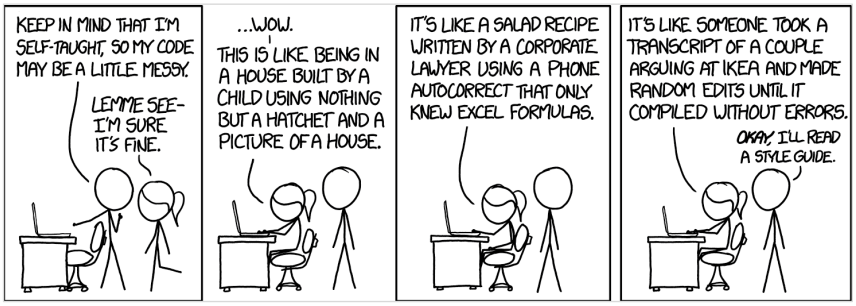
Programming is NOT only about writing code that compiles.

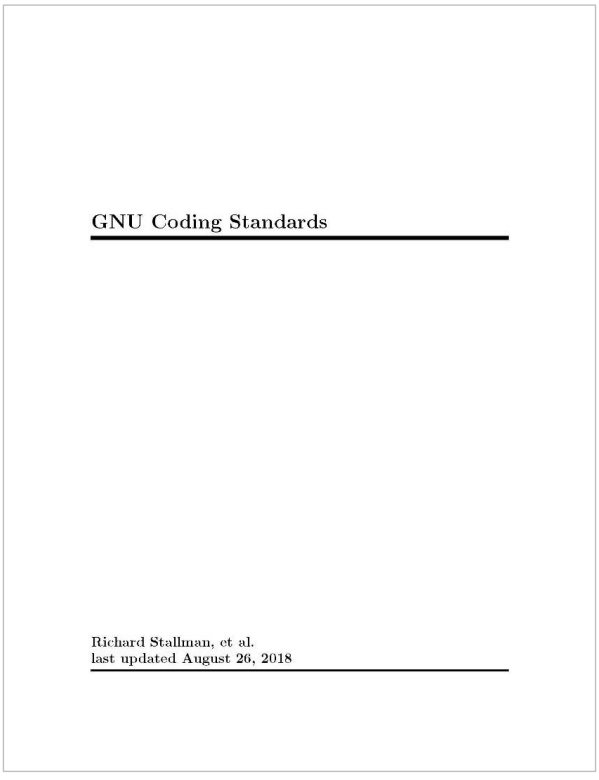
As a customer, you may require use of coding style or conventions.

Some Coding Conventions / Style Guides

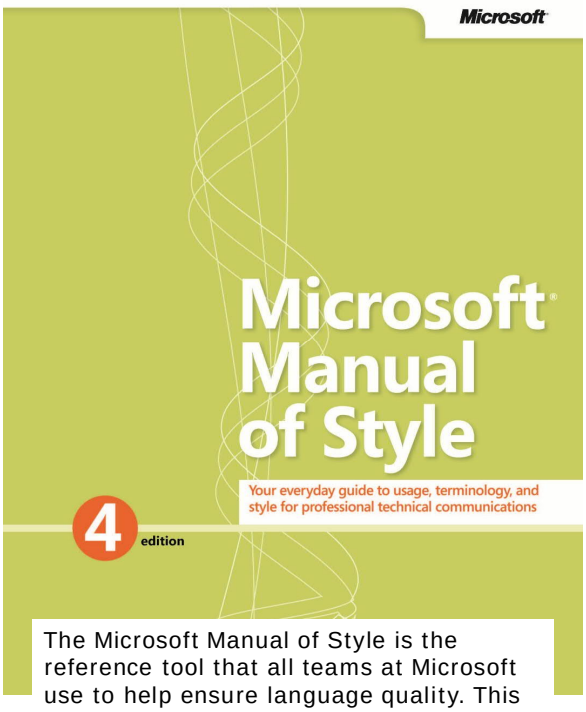
- Google C++ Style Guide
- https://google.github.io/styleguide/cppguide.html
- Google Java Style Guide
- https://google.github.io/styleguide/javaguide.html
- Mozilla codebase coding style
- https://developer.mozilla.org/en-US/docs/Mozilla/Developer_guide/Coding_Style
- Epic games coding standard
- https://docs.unrealengine.com/en-US/Programming/Development/CodingStandard/index.html

Code is read more times than it is written.




GNU Coding Standards

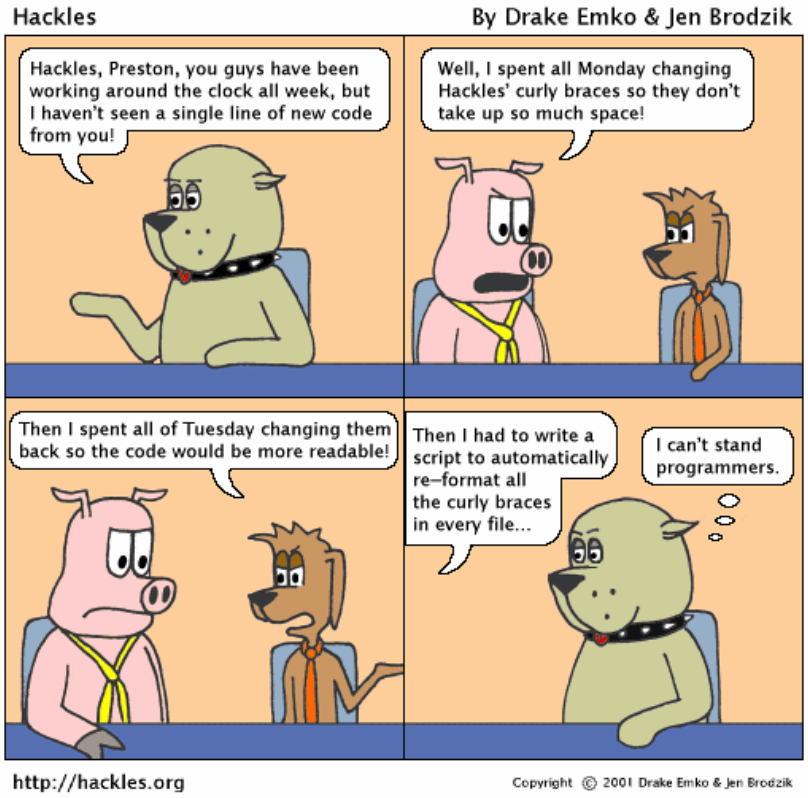
Richard Stallman, et al.
last updated August 26, 2018



The Microsoft Manual of Style is the reference tool that all teams at Microsoft use to help ensure language quality. This guide lays the foundation for the language in our products and services

Tampereen yliopisto
Tampere University

Sometimes there have been programmer ”wars” inside companies about style…



Tampereen yliopisto
Tampere University

Advantages of Implementing Coding Standards in Software Development

- Enhanced Efficiency (e.g. detect problems early)
- Risk of project failure is reduced (e.g. follow rules)
- Minimal Complexity (e.g. develop less complex solution)
- Easy to Maintain (e.g. understandability)
- Bug Rectification (e.g. consistent code, easier to find bugs)
- A Comprehensive Look (e.g. a clear view)
- Cost-Efficient (e.g. reuse).

In nutshell, coding standards play a vital role in any successful software development.

[https://www.multidots.com/importance-of-code-quality-and-coding-standard-in-software-development/]

Tampereen yliopisto
Tampere University

Best practices that are used to write better code

- **Code Comments and Proper Documentation**
- **Use of Indentation**
- **Avoid Commenting on Obvious Things**
- **Grouping Code**
- **Proper and Consistent Scheme for Naming**
- **Principle of DRY** (DRY = Don't Repeat Yourself / DIE = duplication is evil)
- **Deep nesting structure should be avoided**
- **Use short line length**
- **Proper organization of files and folder**
- **OOPs vs. Procedural programming**
- **Open source code readability**
- **Refactoring of code.**

[https://www.multidots.com/importance-of-code-quality-and-coding-standard-in-software-development/]

Tampereen yliopisto
Tampere University

Just another...

Clean code cheat sheet

General rules	Names rules	Use as explanation of intent	Small number of instance variables
Follow standard conventions	Choose descriptive and unambiguous names	Use as clarification of code	Base class should know nothing about their derivatives
Keep it simple stupid. Simpler is always better Reduce complexity as much as possible	Make meaningful distinction	Use as warning of consequences	Better to have many functions than to pass some code into a function to select a behavior
Boy scout rule. Leave the campground cleaner than you found it	Use pronounceable names	**Source code structure**	Prefer non-static methods to static methods
Always find root cause Always look for the root cause of a problem	Use searchable names	Separate concepts vertically	**Tests**
Design rules	Replace magic numbers with named constants	Related code should appear vertically dense	One assert per test
Keep configurable data at high levels	Avoid encodings. Don't append prefixes or type information	Declare variables close to their usage	Readable
Prefer polymorphism to if/else or switch/case	**Functions rules**	Dependent functions should be close	Fast
Separate multi-threading code	Small	Similar functions should be close	Independent
Prevent over-configurability	Do one thing	Place functions in the downward direction	Repeatable
Use dependency injection	Use descriptive names	Keep lines short	**Code smells**
Follow Law of Demeter A class should know only its direct dependencies	Prefer fewer arguments	Don't use horizontal alignment	Rigidity. The software is difficult to change. A small change causes a cascade of subsequent changes
	Have no side effects	Use white space to associate related things and disassociate weakly related	Fragility. The software breaks in many places due to a single change
Understandability tips	Don't use flag arguments. Split method into several independent methods that can be called from the client without the flag	Don't break indentation	Immobility. You cannot reuse parts of the code in other projects because of involved risks and high effort
Be consistent If you do something a certain way, do all similar things in the same way	**Comments rules**	**Objects and data structures**	Needless. Complexity
Use explanatory variables	Always try to explain yourself in code	Hide internal structure	Needless. Repetition
Encapsulate boundary conditions. Boundary conditions are hard to keep track of Put the processing for them in one place	Don't be redundant	Prefer data structures	Opacity. The code is hard to understand
Prefer dedicated value objects to primitive type	Don't add obvious noise	Avoid hybrids structures (half object and half data)	
Avoid logical dependency. Don't write methods which works correctly depending on something else in the same class	Don't use closing brace comments	Should be small	'**Clean code**' by Robert C. Martin summary by Wojtek Lukaszuk
Avoid negative conditionals	Don't comment out code Just remove	Do one thing	

CI = continuous integration

CD = continuous deployment

CI

Continuous Integration is the practice of merging code changes into a shared repository several times a day in order to release a product version at any moment. This requires an integration procedure which is reproducible and automated.

[Agile Alliance]

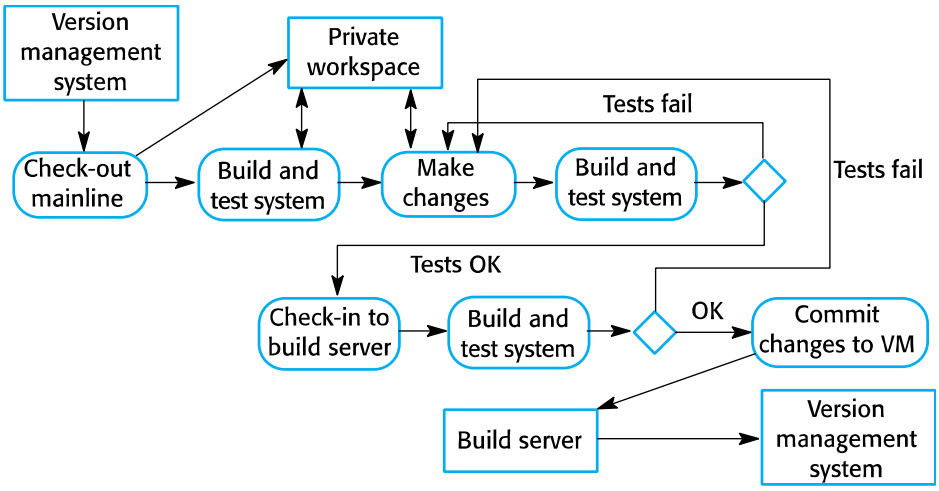
continuous integration

Teams practicing continuous integration seek two objectives:

- minimize the duration and effort required by each integration episode
- be able to deliver a product version suitable for release at any moment
- In practice, this dual objective requires an integration procedure which is reproducible at the very least, and largely automated. This is achieved through version control tools, team policies and conventions, and tools specifically designed to help achieve continuous integration.

[Agile Alliance]

Continuous integration



CD

Continuous deployment aims to reduce the time elapsed between writing a line of code and making that code available to users in production. To achieve continuous deployment, the team relies on infrastructure that automates and instruments the various steps leading up to deployment, so that after each integration successfully meeting these release criteria, the live application is updated with new code.

[Agile Alliance]

Bug report(ing)

What are the qualities of a good software bug report?

Anyone can write a bug report. But not everyone can write a effective bug report. You should be able to distinguish between average bug report and a good bug report. How to distinguish a good or bad bug report? It's simple, apply following characteristics and techniques to report a bug.

1) Having clearly specified bug number:
2) Reproducible:
3) Be Specific:

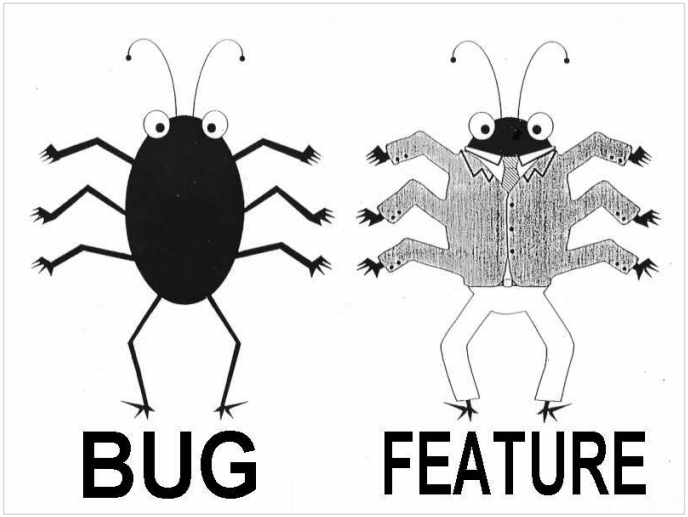
http://www.softwaretestinghelp.com/

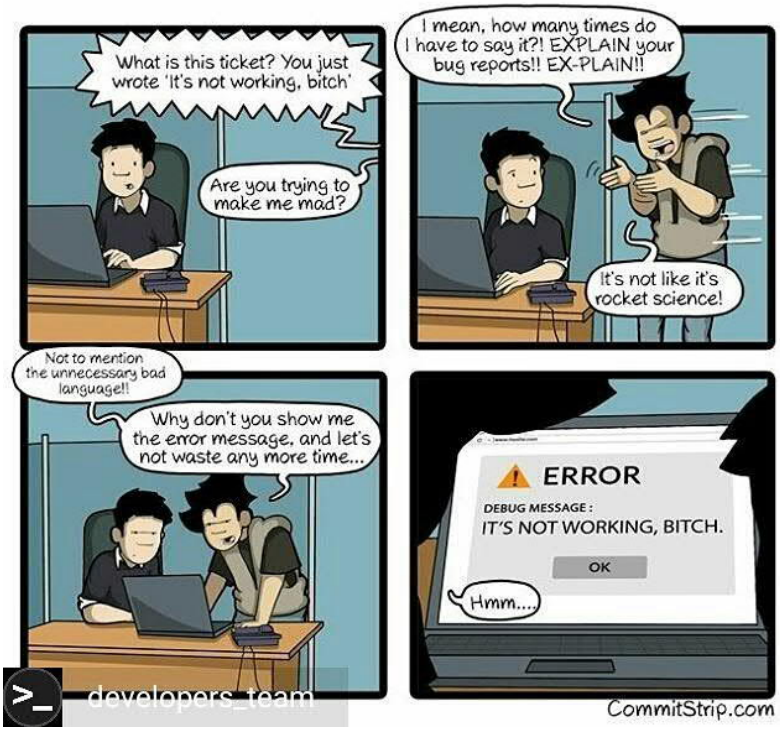
How to Report a Bug?

Use following simple Bug report template:

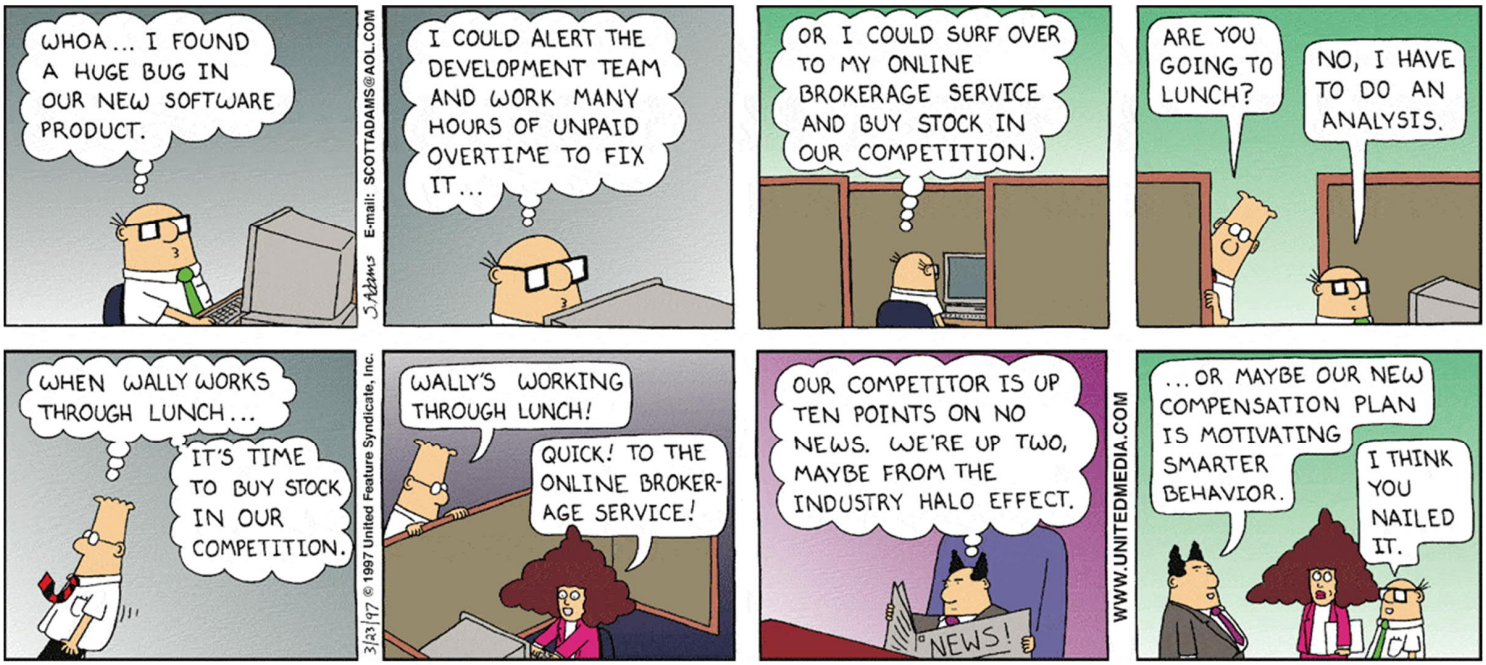
- Reporter: Your name and email address.
- Product: In which product you found this bug.
- Version: The product version if any.
- Component: These are the major sub modules of the product.
- Platform: Mention the hardware platform where you found this bug.
- Operating system: Mention all operating systems where you found the bug.
- Priority: When bug should be fixed? Priority is generally set from P1 to P5, P1 as highest.
- Severity: This describes the impact of the bug.
  Types of Severity:
  - Blocker: No further testing work can be done.
  - Critical: Application crash, Loss of data.
  - Major: Major loss of function.
  - Minor: minor loss of function.
  - Trivial: Some UI enhancements.
  - Enhancement: Request for new feature or some enhancement in existing one.
- Status:
- Assign To:
- URL:
- Summary:
- Description:

http://www.softwaretestinghelp.com/



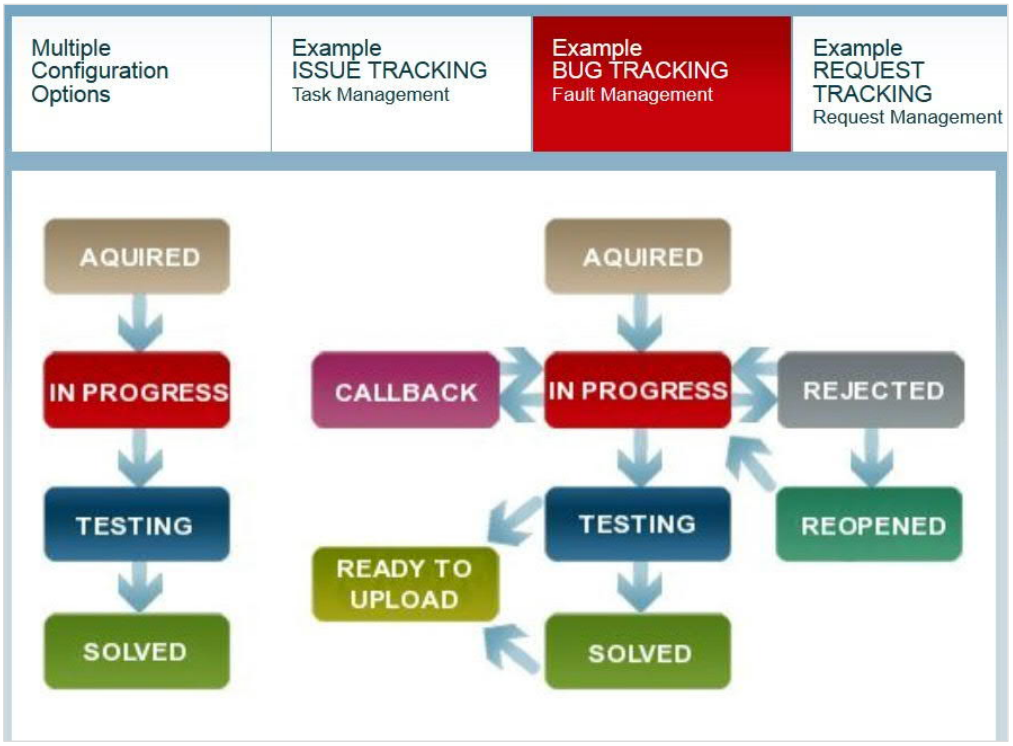


Remember ethics in sw dev and testing



Tampereen yliopisto
Tampere University

One example of bug tracking process



http://www.woodpecker-it.com/

Tampereen yliopisto
Tampere University

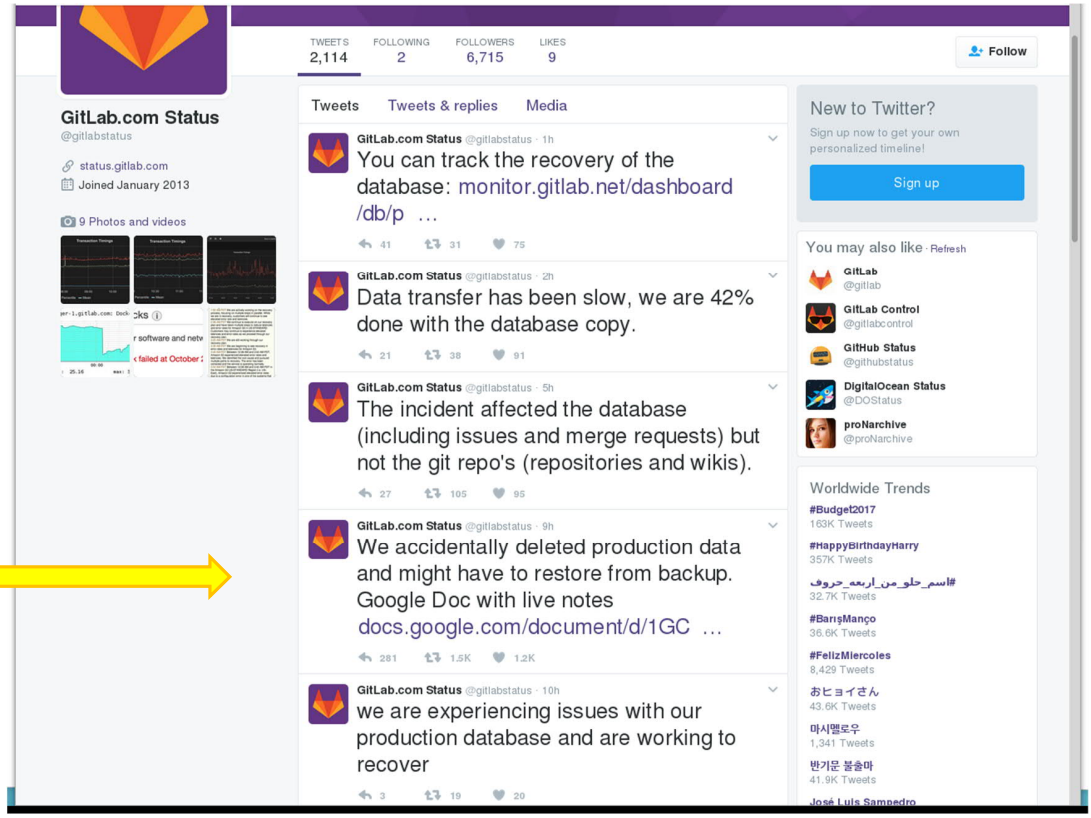
Some open source bug databases

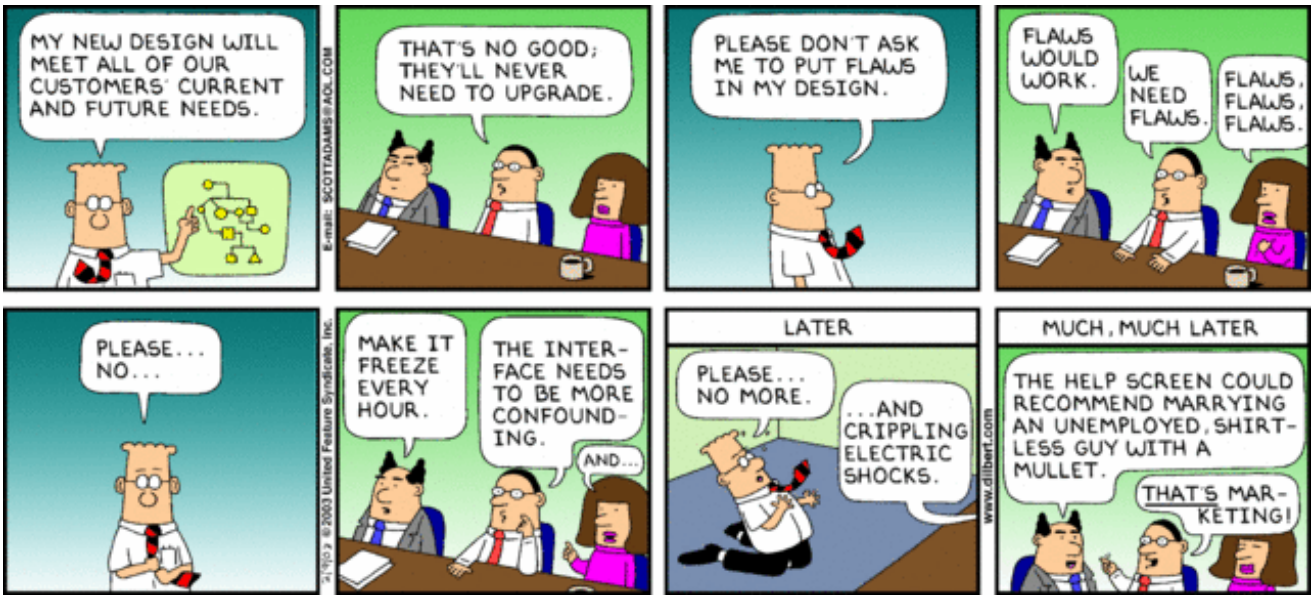
- Bugzilla http://www.bugzilla.org
- Mantis http://www.mantisbt.org
- Trac http://trac.edgewall.org/
- Redmine http://www.redmine.org/
- Request tracker http://bestpractical.com/rt/
- EventNum https://github.com/eventum
- Fossil http://www.fossil-scm.org
- The Bug Genie http://www.thebuggenie.com/
- WebIssues http://webissues.mimec.org/

And many more exists…

01.02.2017.

Luckily they had backups…





Bug bounty
White-hat hackers

Bug Bounty (some sites)

hackr.fi Bug Bounty, CROWDSOURCED SECURITY TESTING

Hackrfi enables and manages bug bounty programs. With us you'll create a useful bug bounty program and get great support through its life cycle.

hackerone.com Internet Bug Bounty, The Most Trusted Hacker-Powered Security Platform

bugcrowd.com PUBLIC BUG BOUNTY LIST

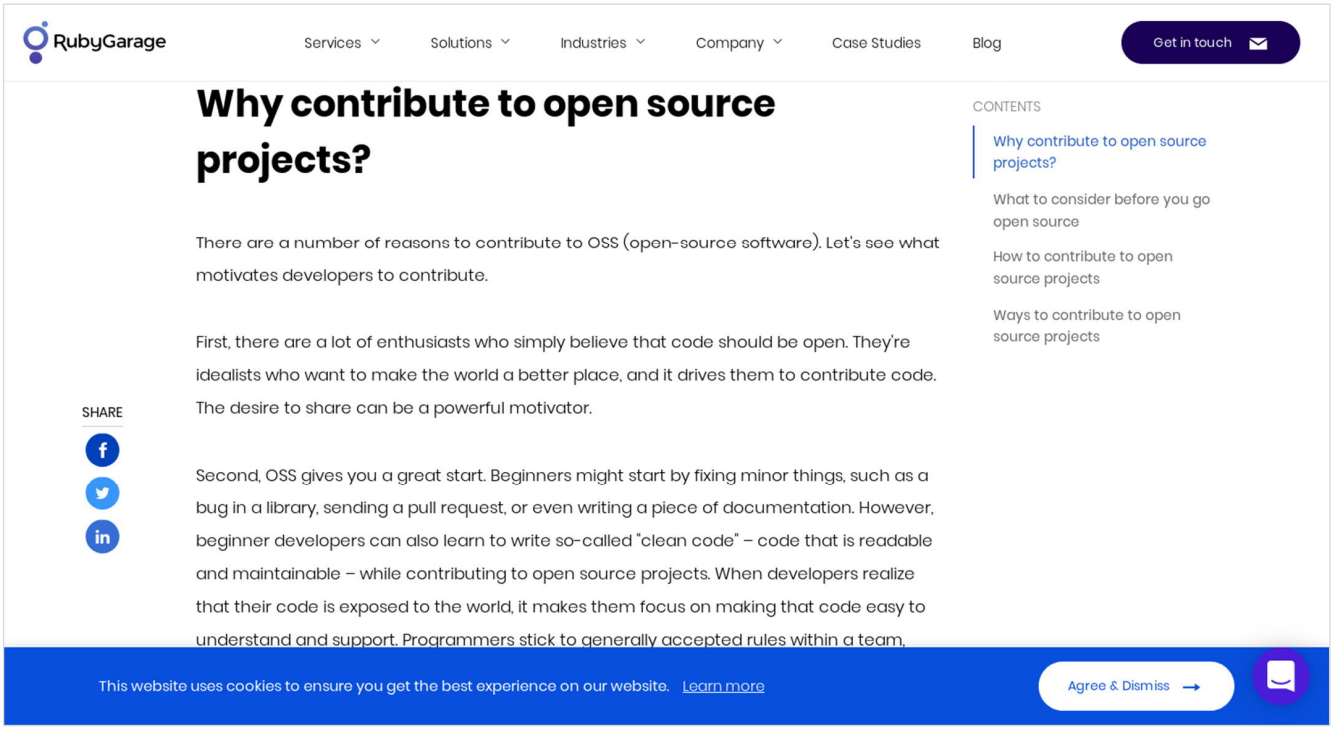
The most comprehensive, up to date crowdsourced list of bug bounty and security disclosure programs from across the web curated by the hacker community.

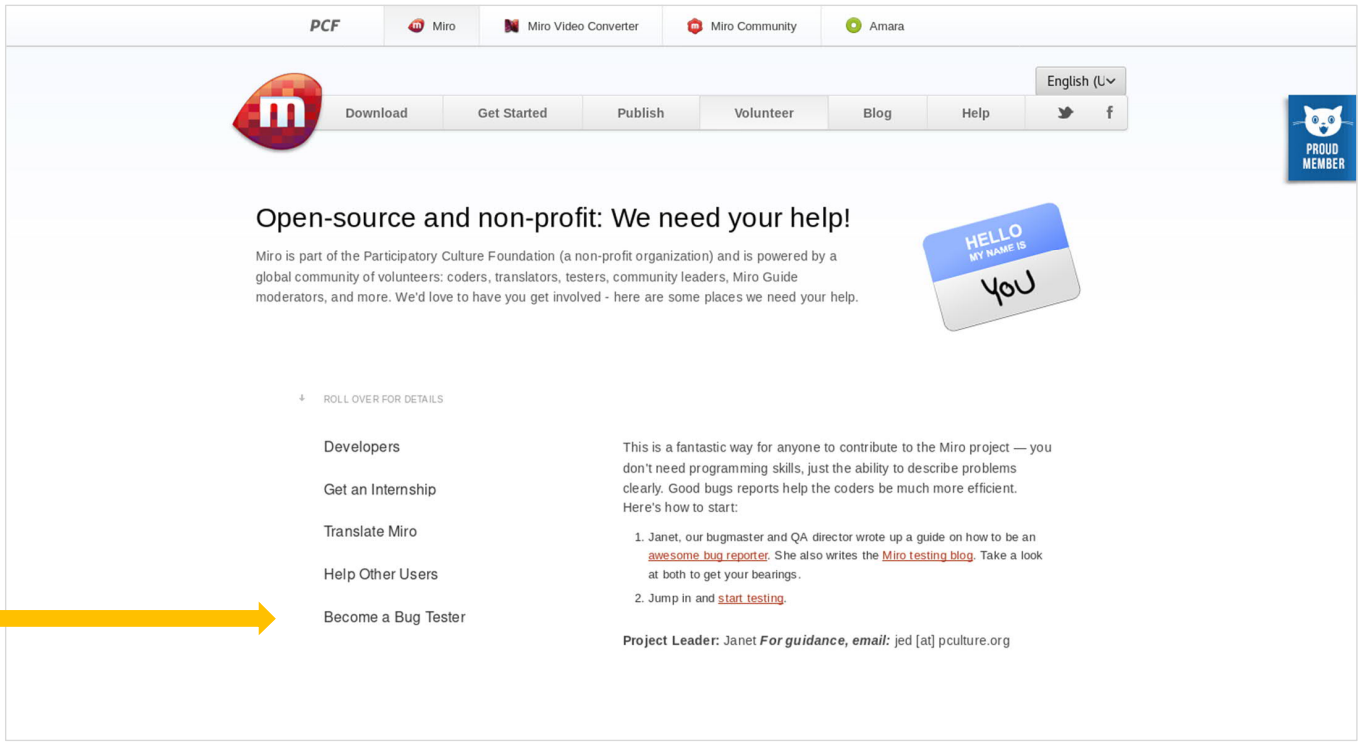
Highest known Bug Bounty reward in Finland: 18000 euros.

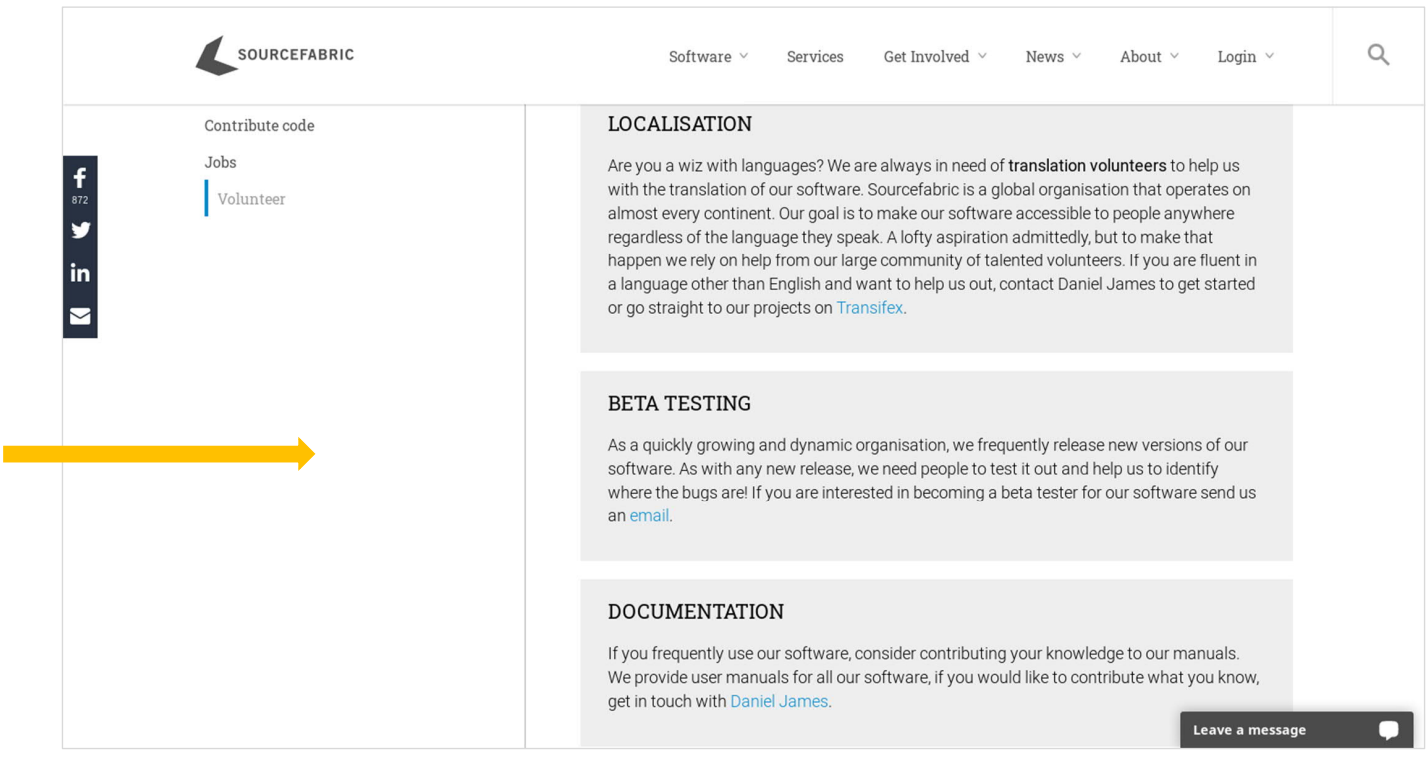
Tampereen yliopisto
Tampere University

Contribute to open source projects (?)

Tampereen yliopisto
Tampere University

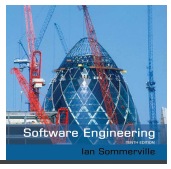



PCF
Miro
Miro Video Converter
Miro Community
Amara
English (U
Download
Get Started
Publish
Volunteer
Blog
Help
PROUD MEMBER
Open-source and non-profit: We need your help!
Miro is part of the Participatory Culture Foundation (a non-profit organization) and is powered by a global community of volunteers: coders, translators, testers, community leaders, Miro Guide moderators, and more. We'd love to have you get involved - here are some places we need your help.
HELLO MY NAME IS YOU
ROLL OVER FOR DETAILS
Developers
Get an Internship
Translate Miro
Help Other Users
Become a Bug Tester
This is a fantastic way for anyone to contribute to the Miro project — you don't need programming skills, just the ability to describe problems clearly. Good bugs reports help the coders be much more efficient. Here's how to start:
1. Janet, our bugmaster and QA director wrote up a guide on how to be an awesome bug reporter. She also writes the Miro testing blog. Take a look at both to get your bearings.
2. Jump in and start testing.
Project Leader: Janet For guidance, email: jed [at] pculture.org


SOURCEFABRIC
Software
Services
Get Involved
News
About
Login
Contribute code
Jobs
Volunteer
872
LOCALISATION
Are you a wiz with languages? We are always in need of translation volunteers to help us with the translation of our software. Sourcefabric is a global organisation that operates on almost every continent. Our goal is to make our software accessible to people anywhere regardless of the language they speak. A lofty aspiration admittedly, but to make that happen we rely on help from our large community of talented volunteers. If you are fluent in a language other than English and want to help us out, contact Daniel James to get started or go straight to our projects on Transifex.
BETA TESTING
As a quickly growing and dynamic organisation, we frequently release new versions of our software. As with any new release, we need people to test it out and help us to identify where the bugs are! If you are interested in becoming a beta tester for our software send us an email.
DOCUMENTATION
If you frequently use our software, consider contributing your knowledge to our manuals. We provide user manuals for all our software, if you would like to contribute what you know, get in touch with Daniel James.
Leave a message

TDD = test driven development

Test-driven development (TDD)



- **Test-driven development** (**TDD**) is an approach to program development in which you inter-leave testing and code development.
- **Tests are written before code** and 'passing' the tests is the critical driver of development.
- You develop code incrementally, along with a test for that increment. You don't move on to the next increment until the code that you have developed passes its test.
- TDD was introduced as part of agile methods such as Extreme Programming. However, it can also be used in plan-driven development processes.

https://www.agilealliance.org/glossary/tdd

“Test-driven development” refers to a style of programming in which three activities are tightly interwoven: coding, testing (in the form of writing unit tests) and design (in the form of refactoring).

It can be succinctly described by the following set of rules:

- write a “single” unit test describing an aspect of the program
- run the test, which should fail because the program lacks that feature
- write “just enough” code, the simplest possible, to make the test pass
- “refactor” the code until it conforms to the simplicity criteria
- repeat, “accumulating” unit tests over time.

https://www.agilealliance.org/glossary/tdd

Expected Benefits

- many teams report significant reductions in defect rates, at the cost of a moderate increase in initial development effort
- the same teams tend to report that these overheads are more than offset by a reduction in effort in projects’ final phases
- although empirical research has so far failed to confirm this, veteran practitioners report that TDD leads to improved design qualities in the code, and more generally a higher degree of “internal” or technical quality, for instance improving the metrics of cohesion and coupling.

https://www.agilealliance.org/glossary/tdd

Common Pitfalls

Typical individual mistakes include:

- forgetting to run tests frequently
- writing too many tests at once
- writing tests that are too large or coarse-grained
- writing overly trivial tests, for instance omitting assertions
- writing tests for trivial code, for instance accessors.

Typical team pitfalls include:

- partial adoption – only a few developers on the team use TDD
- poor maintenance of the test suite – most commonly leading to a test suite with a prohibitively long running time
- abandoned test suite (i.e. seldom or never run) – sometimes as a result of poor maintenance, sometimes as a result of team turnover.

Version control

Versioning terminology… it depends…

Version (terminology varies…)

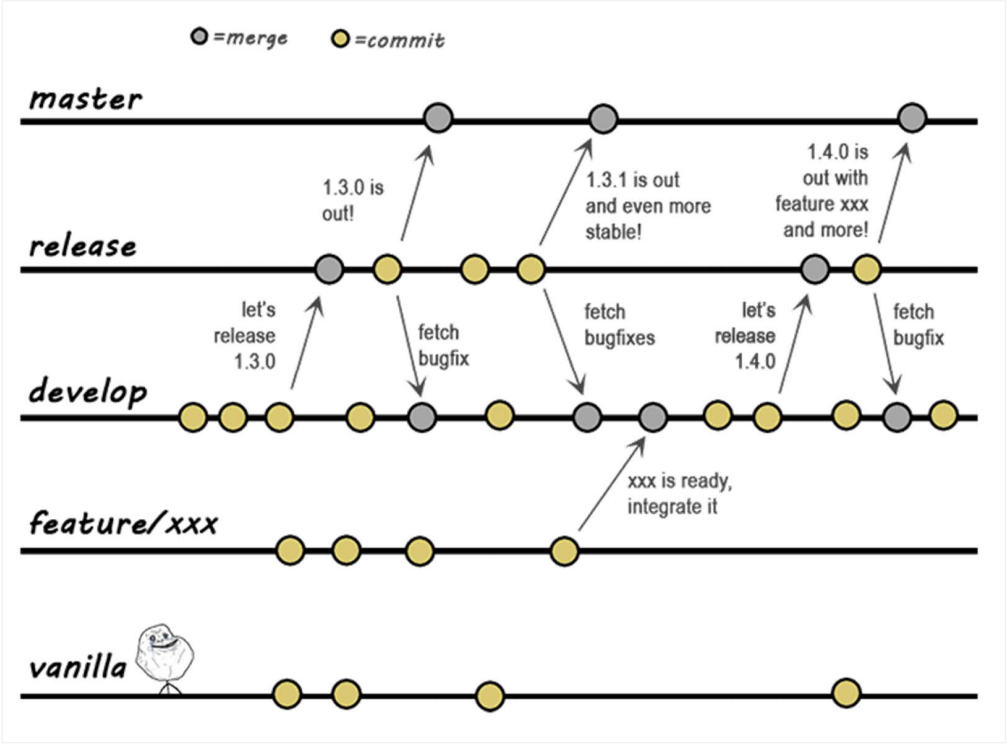
Revision (nothing at ISO 24765 std; as you see, terminology varies…)

- “In software development a revision is seen as a major release of the software that will introduce new features and functionality to the application. In the engineering world revisions are done to show changes to a design so that the changes are documented for the entire design team and any other individual who uses the data.”
- ”A new revision contains minor changes, usually only corrections. A new version has major changes, which may include additions and reorganization of the contents.”

Release (terminology varies…)

Build “a build is usually a version of software in pre-release format that is used only by the software development company”.

Vanilla = unfeatured version (plain, basic). Karvalakkiversio.



[GitHub]

Tampereen yliopisto
Tampere University

Versions

Peer review at version control tool;

- Merge request (GitLab)
- Pull request (GitHub)

By the way, about version numbers. What comes after version 1.9 ?

- 2.0
- 1.10
- 1.9.1
- 1.91
- 1.9b ?

Tampereen yliopisto
Tampere University

Build... an internal version

3.431 , build

1. operational version of a system or component that incorporates a specified subset of the capabilities that the final product will provide

2. process of generating (archiving) an executable and testable system from source versions or baselines

3. to perform the steps required to produce an instance of the product

Note 1 to entry: In software, this means processing source files to derive target files. In hardware, this means assembling a physical object. The build needs to compile and link the various versions in the correct order. The build tools can be integrated into a configuration management tool.

[ISO 24765:2017]

Release… particular version deployed

3.3382 , release

1. particular version of a configuration item that is made available for a specific purpose

2. collection of new or changed configuration items that are tested and introduced into a live environment together

3. collection of one or more new or changed configuration items deployed into the live environment as a result of one or more changes

4. software version that is made formally available to a wider community

5. Delivered version of an application which includes all or part of an application

6. set of grouped change requests, established in the Application Change Management Process, which are designed, developed, tested, and deployed as a cohesive whole.

cf. version.

[ISO 24765:2017]

Version… initial release

3.4545 , version

1. initial release or re-release of a computer software configuration item, associated with a complete compilation or recompilation of the computer software configuration item

2. initial release or complete re-release of a document, as opposed to a revision resulting from issuing change pages to a previous release

3. operational software product that differs from similar products in terms of capability, environmental requirements, and configuration

4. identified instance of a configuration item

5. identified instance of an item

6. unique string of number and letter values indicating a unique revision of an item

cf. release

Note 1 to entry: Versions often identify revisions of software that provide unique functionality or fixes. A version typically has multiple parts, such as a major version, indicating large changes in functionality or user interface changes, and a minor version, indicating smaller changes in functionality or user interface changes.

[ISO 24765:2017]

Tampereen yliopisto
Tampere University

baseline [ISO 24765:2017]

3.339 , baseline

1. formally approved version of a configuration item, regardless of media, formally designated and fixed at a specific time during the configuration item's life

2. specification or product that has been formally reviewed and agreed upon, that thereafter serves as the basis for further development, and that can be changed only through formal change control procedures

3. agreement or result designated and fixed at a given time, from which changes require justification and approval

4. snapshot of the state of a service or individual configuration items at a point in time

5. formally controlled and maintained set of data that serves as the basis for defining change

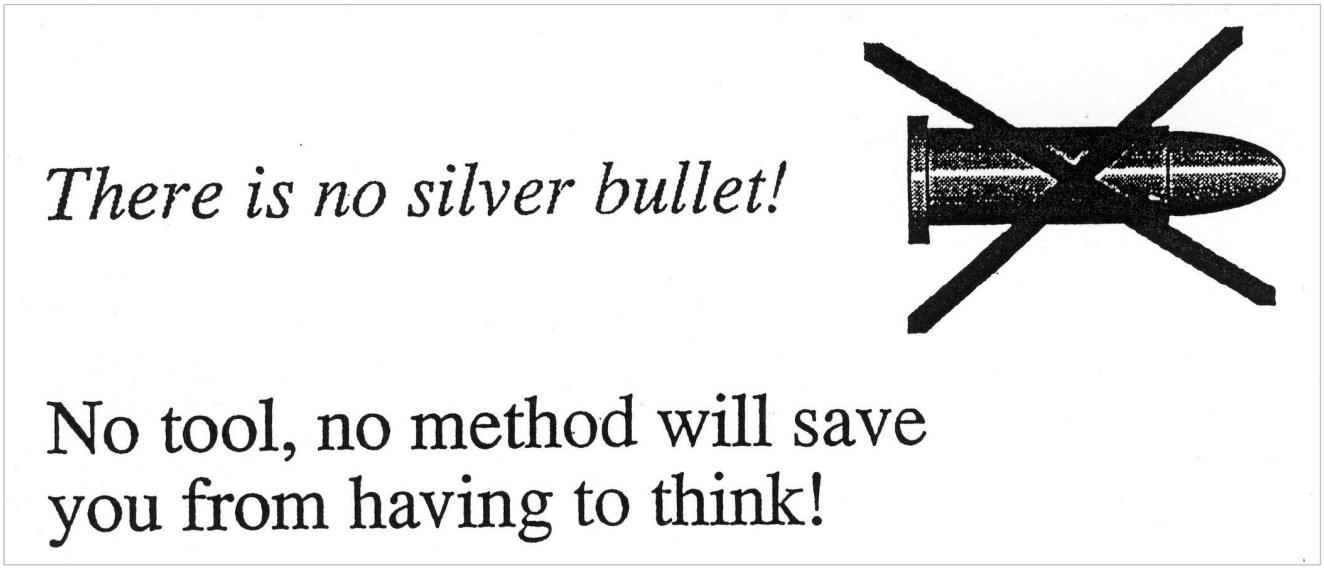
6 [verb] to establish and approve a set of data

7. the approved version of a work product that can be changed only through formal change control procedures and is used as a basis for comparison

8. approved version of a configuration item, regardless of media, formally designated and fixed at a specific time during the configuration item's life cycle.

Note 1 to entry: Some baselines are project deliverables, while others provide the basis for further work. A baseline, together with all approved changes to the baseline, represents the current approved configuration. The term is thus used to refer to a particular version of a software configuration item that has been agreed on, e.g., as a stable base for further development or to mark a specific project milestone. In either case, any new baseline is agreed through the project's agreed change control procedures.

Tampereen yliopisto
Tampere University

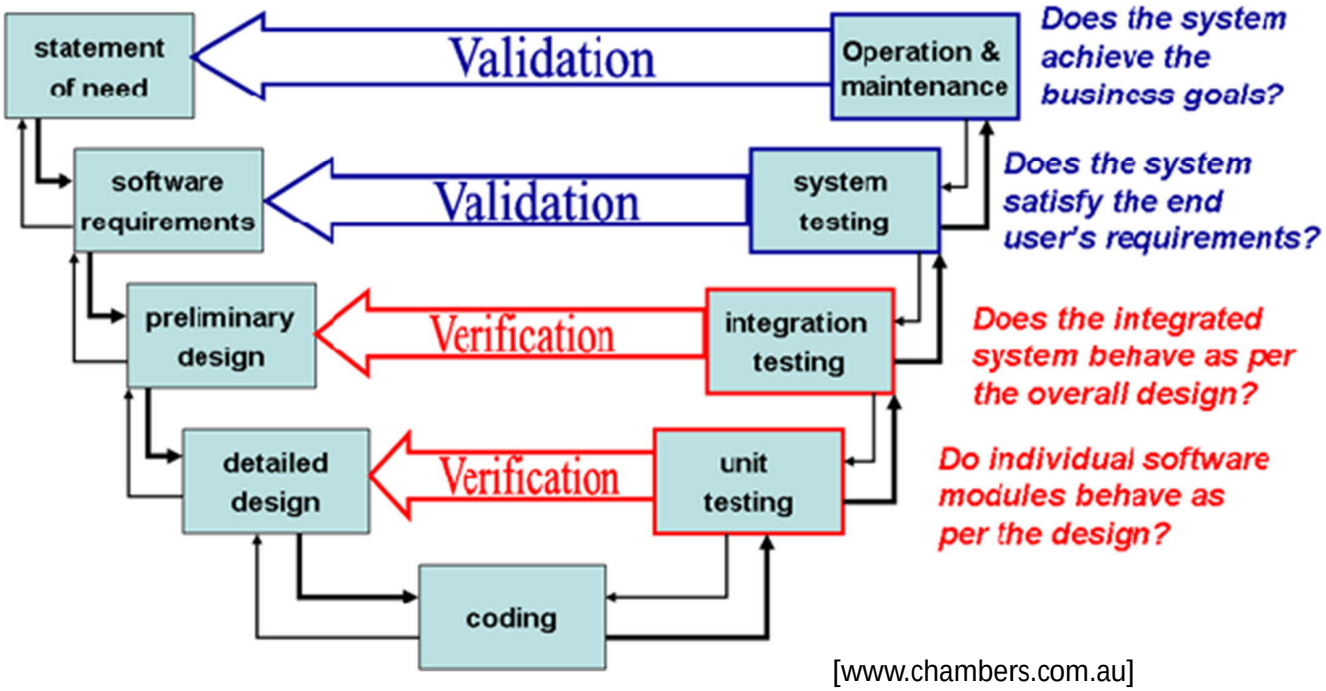


V & V = verification and validation

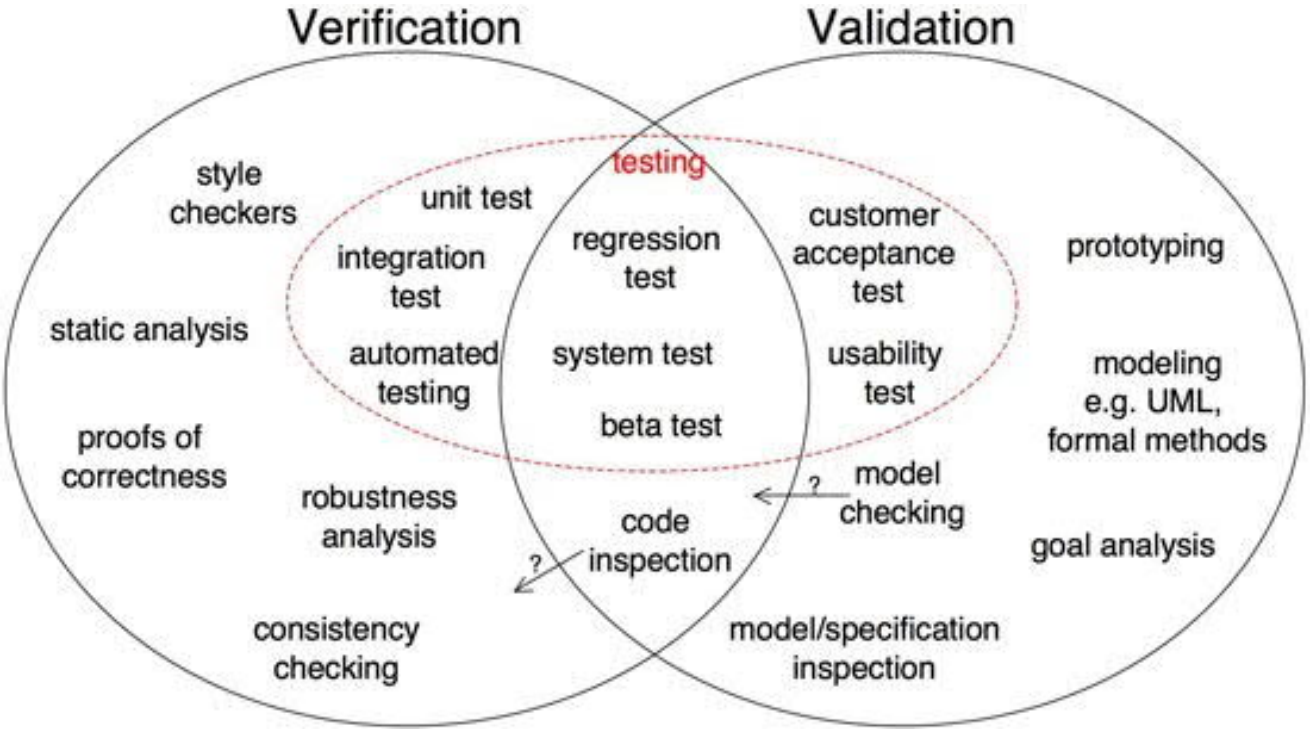
V&V, verification and validation

- verification, are we building the product right ?
- validation, are we building the right product ?

Dynamic Testing



[www.chambers.com.au]



[www.easterbrook.ca]

Tampereen yliopisto
Tampere University

[ISO 24765:2018]

3.4538 , verification (FI: todentaminen)

1. confirmation, through the provision of objective evidence, that specified requirements have been fulfilled

2. the evaluation of whether or not a product, service, or system complies with a regulation, requirement, specification, or imposed condition. It is often an internal process

3. process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase

4. process of providing objective evidence that the system, software, or hardware and its associated products conform to requirements (e.g., for correctness, completeness, consistency, and accuracy) for all life cycle activities during each life cycle process (acquisition, supply, development, operation, and maintenance), satisfy standards, practices, and conventions during life cycle processes, and successfully complete each life cycle activity and satisfy all the criteria for initiating succeeding life cycle activities.

cf. validation

Tampereen yliopisto
Tampere University

[ISO 24765:2018]

3.4500 , validation (FI: kelpoistaminen)

1. confirmation, through the provision of objective evidence, that the requirements for a specific intended use or application have been fulfilled

2. process of providing evidence that the system, software, or hardware and its associated products satisfy requirements allocated to it at the end of each life cycle activity, solve the right problem (e.g., correctly model physical laws, implement business rules, and use the proper system assumptions), and, and satisfy intended use and user needs

3. In a life cycle context, the set of activities ensuring and gaining confidence that a system is able to accomplish its intended use, goals and objectives

4. process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements

5. the assurance that a product, service, or system meets the needs of the customer and other identified stakeholders. It often involves acceptance and suitability with external customers.

CM = configuration management

Tampereen yliopisto
Tampere University

CM = configuration management

3.779 , configuration management (CM)

1. discipline applying technical and administrative direction and surveillance to: identify and document the functional and physical characteristics of a configuration item, control changes to those characteristics, record and report change processing and implementation status, and verify compliance with specified requirements

2. technical and organizational activities, comprising configuration identification, control, status accounting and auditing

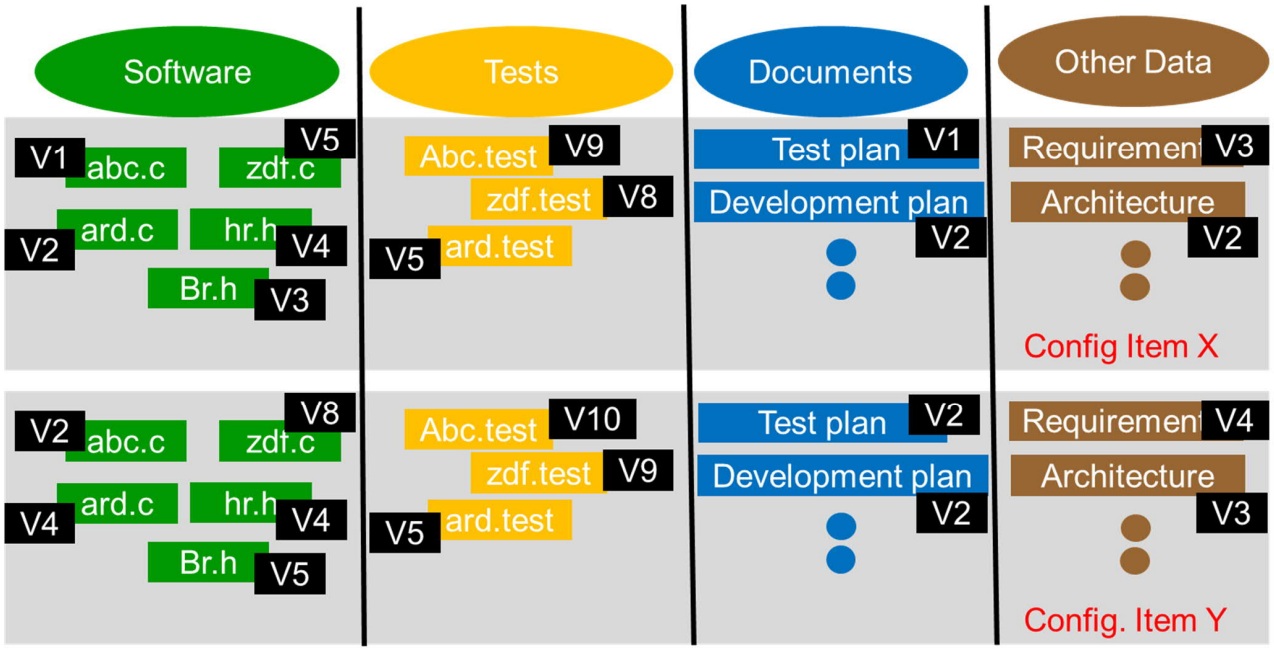
3. coordinated activities to direct and control the configuration

cf. baseline, change management, configuration identification, configuration control, configuration status accounting, configuration audit.

[ISO 24765:2017]

Software Configuration Management

You have better to have some good system for managing software configurations



[http://blog.heicon-ulm.de/configuration-management-a-challenging-task/]

Tampereen yliopisto
Tampere University

Tampereen yliopisto
Tampere University

Highlights - What to remember

- quality is NOT something you can add to a product later, it must be in the development process from the beginning
- outsourcing sounds great, but plan and implement it carefully
- software testing is worth automation, e.g. unit tests (CI pipeline)
- ”ad-hoc” testing without planning (nor documenting) is quite useless
- from testing there should be some evidence, too (e.g. test log or diary)
- at a system test you can not test every combination of users’ actions (e.g. at www applications and games), you may use e.g. exploratory testing with some ”story” about what to do
- terminology is not coherent… it varies… depends on adopted process…
- there are many tools to help on e.g. static code analysis, version control and configuration management.

Tampereen yliopisto
Tampere University

Tampereen yliopisto
Tampere University

Now the additional L8 extra slides are here

No time to show these at lectures, but otherwise good to know, at least if you are a major reader.

Tampereen yliopisto
Tampere University

Now the additional L8 extra slides are here

No time to show these at lectures, but otherwise good to know, at least if you are a major reader.

Tampereen yliopisto
Tampere University

Now the additional L8 extra slides are here

No time to show these at lectures, but otherwise good to know, at least if you are a major reader.

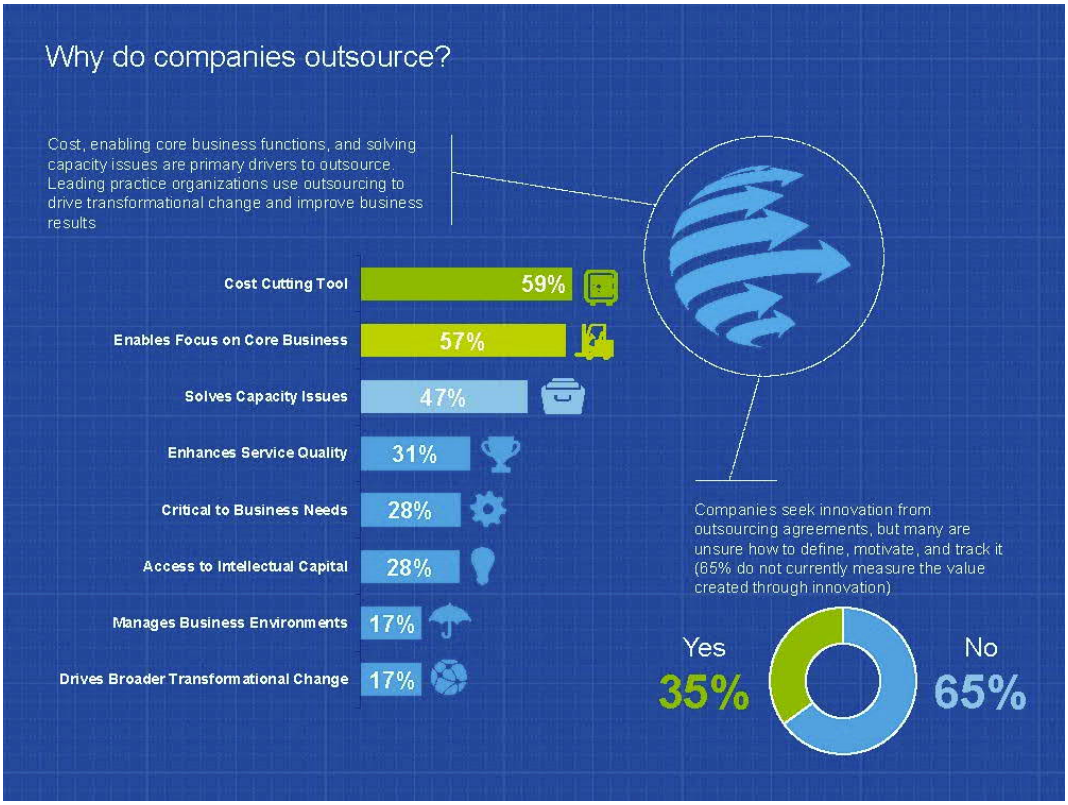
Tampereen yliopisto
Tampere University

Now the additional L8 extra slides are here

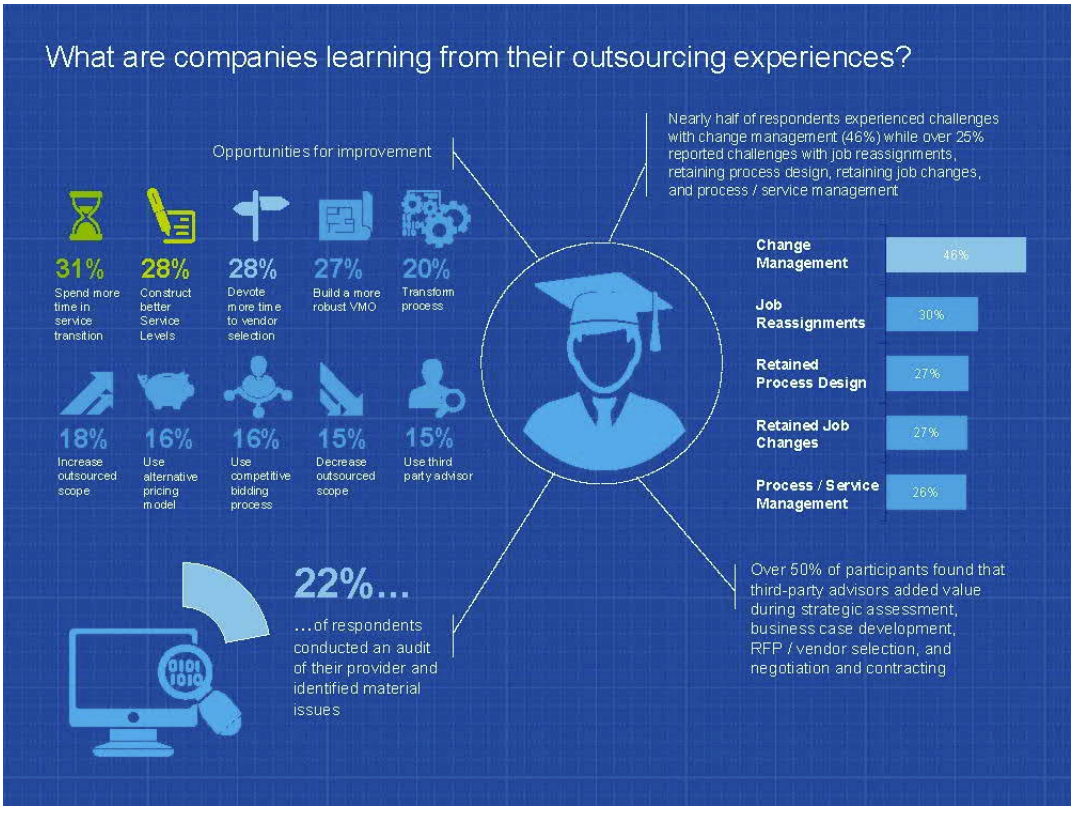
No time to show these at lectures, but otherwise good to know, at least if you are a major reader.

Tampereen yliopisto
Tampere University

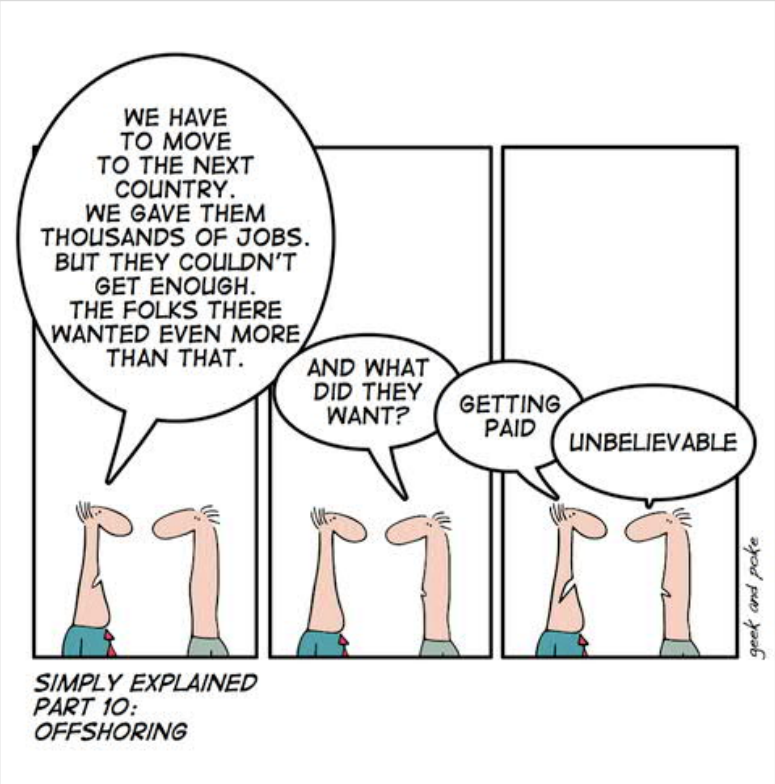
GSD = Global software development



[www.deloitte.com , 2016]



[www.deloitte.com , 2016]



Maintenance

Tampereen yliopisto
Tampere University

What is Application Maintenance, 1

Due to evolving customer expectations, the fight to survive in an existing market, and technological advancements, modifying and implementing new strategies is critical in maintaining sustainability and staying competitive. Every competitive business needs to constantly enhance and manage the IT solutions that have been developed in order to stay relevant and meet the wavering needs of users.

Contrary to popular belief, application maintenance is not just about fixing defects, but modifying a software product after delivery to correct faults, as well as to improve performance. Application maintenance and enhancement to existing applications begin with a thorough study of existing applications to identify areas of improvement.

Trends in the application development and maintenance landscape:

- Security practices will no longer be implemented at a post-development stage but as code
- Cloud implementations will become a popular way to meet the needs of end-users
- Automation will enable organizations to consolidate tools so they don't have to custom-build deployments.
- Innovative tools and technologies are readily available and are cost-optimized for an application landscape.

[https://synoptek.com/insights/it-blogs/application-development-maintenance/]

Tampereen yliopisto
Tampere University

What is Application Maintenance, 2

A few tips:

- Be as clear as possible as to what your requirements for your application are
- Thoroughly understand the services offered by application development companies and identify the right partner if you're using a partner
- Evaluate the various development platforms and choose the one that best fits the needs of your business
- Make sure to embed processes that focus on continuous improvements and iterations to add new features and/or fix bugs
- When developing your application, make security your top priority
- Regularly update and test your application to deliver improved and better performance, high security, and a bug-free, seamless user experience.

[https://synoptek.com/insights/it-blogs/application-development-maintenance/]

Tampereen yliopisto
Tampere University

QA = quality assurance

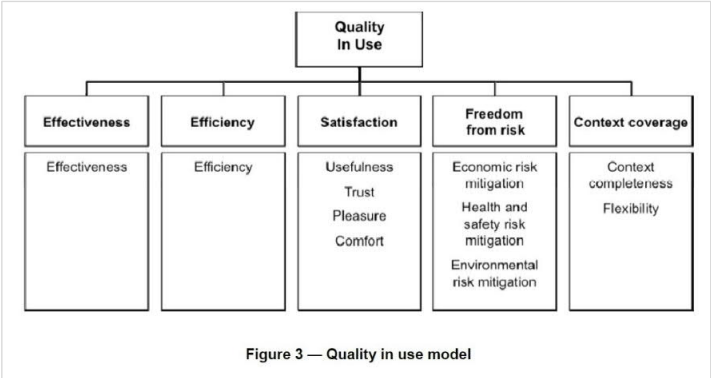


Figure 3 — Quality in use model

ISO 25010

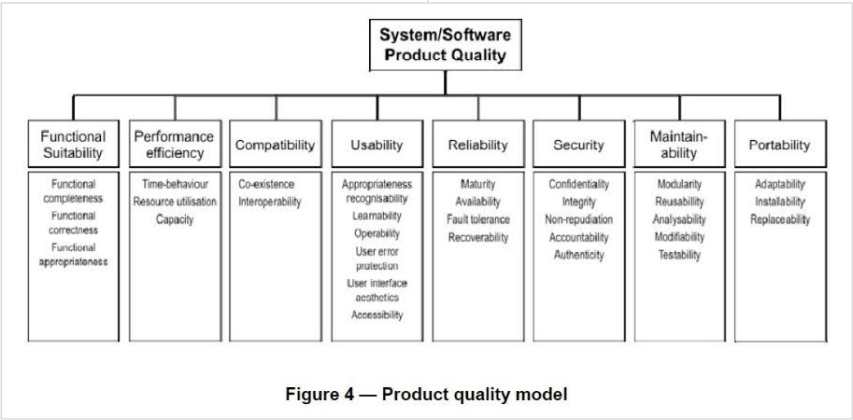


Figure 4 — Product quality model

SQA , SQC

• Software Quality Assurance – Software Quality Assurance (SQA) is a set of activities to ensure the quality in software engineering processes that ultimately result in quality software products. The activities establish and evaluate the processes that produce products. It involves process-focused action.

• Software Quality Control – Software Quality Control (SQC) is a set of activities to ensure the quality in software products. These activities focus on determining the defects in the actual products produced. It involves product-focused action.

[Tutorialspoint]

Quality handbook

Many organisations have some kind of quality handbook, which is a collection of ”best practices”.

If you want to get a quality sertification, you have better to have a quality handbook.

In SMEs (small and medium size enterprises) just quality guidelines may be enough.

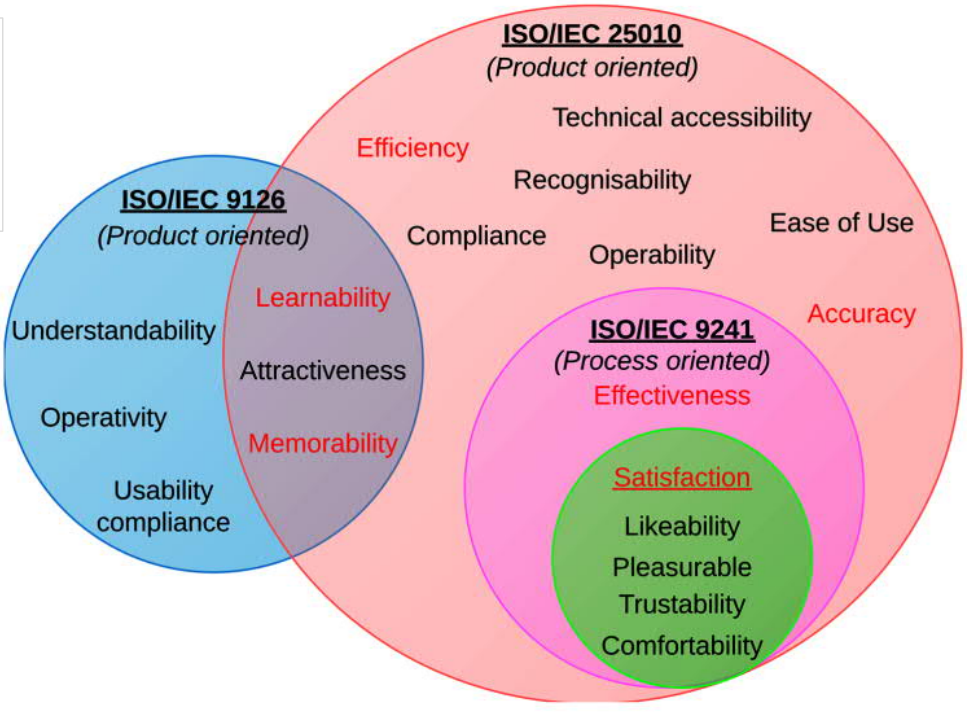
Quality handbook makes sure that tools and practices are the same within the whole organisation.

When making a Quality Handbook, it is advisable to use several standards as checklists; e.g. ISO 10000-family, ISO 25000-family, and PMBoK.

Encyclopedia of Information Science and Technology, Fourth Edition

Mehdi Khosrow-Pour
Information Resources Management Association, USA

Figure 1. Usability definitions according to ISO/IEC 9241-11, ISO/IEC 9126 and ISO/IEC 25010



There are many classifications of software quality factors

[www.compact.nl]

There are many classifications of quality factors

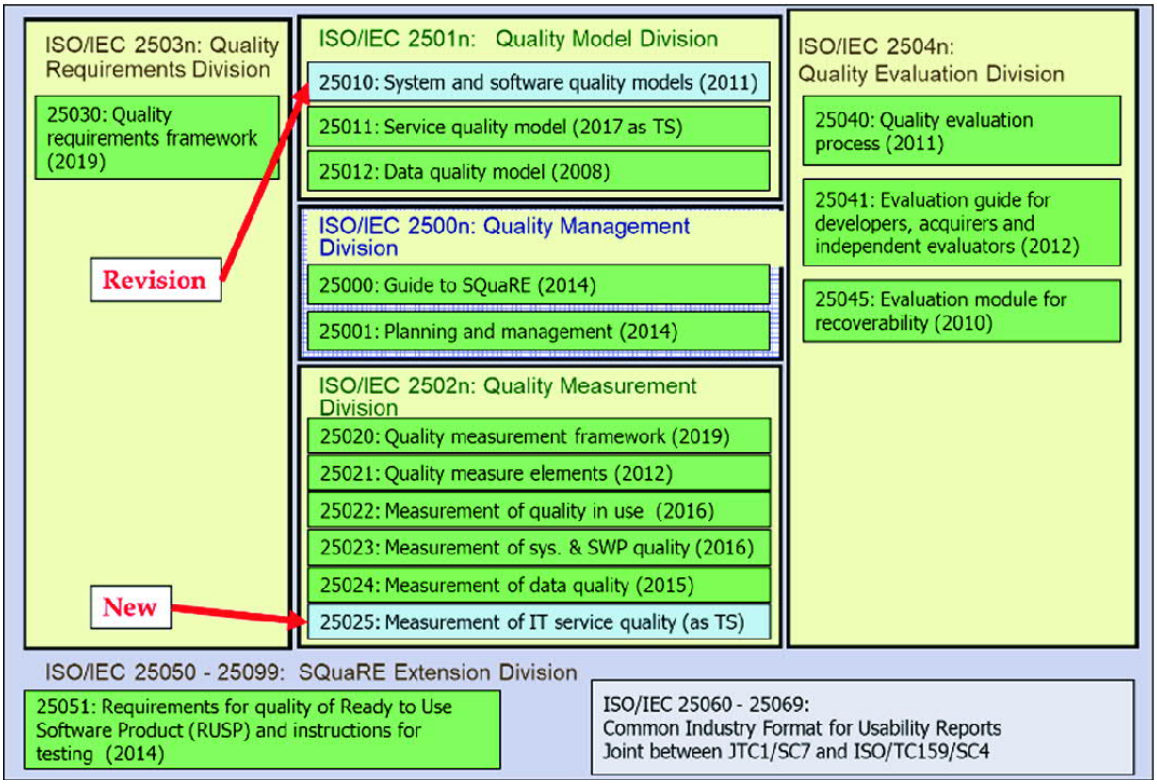
Sw quality factors

McCall's Factor Model

- This model classifies all software requirements into 11 software quality factors. The 11 factors are grouped into three categories – product operation, product revision, and product transition factors.

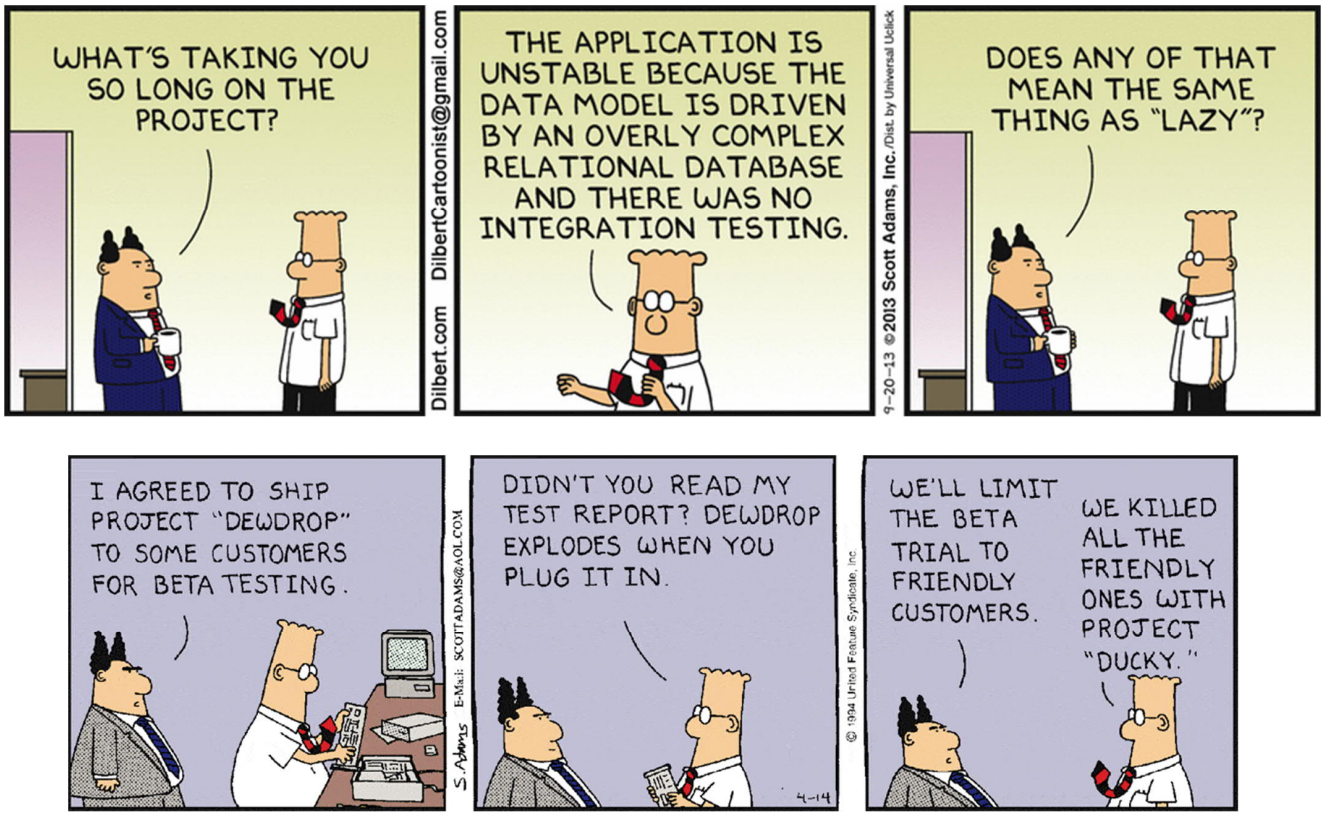
- Product operation factors – Correctness, Reliability, Efficiency, Integrity, Usability.
- Product revision factors – Maintainability, Flexibility, Testability.
- Product transition factors – Portability, Reusability, Interoperability.

[Tutorialspoint]



[Usability of Software–Intensive Systems from Developers' Point of View, 2020]

Inspections and reviews



Distributed reviews

- The processes suggested for reviews assume that the review team has a face-to-face meeting to discuss the software or documents that they are reviewing.
- However, **project teams are now often distributed**, sometimes across countries or continents, so it is impractical for team members to meet face to face.
- Remote reviewing can be supported using shared documents where each review team member can annotate the document with their comments.

Program inspections

- These are **peer reviews** where engineers examine the source of a system with the aim of discovering anomalies and defects.
- **Inspections do not require execution of a system so may be used before implementation.**
- They may be applied to any representation of the system (requirements, design,configuration data, test data, etc.).
- They have been shown to be an **effective technique** for discovering program errors.

Inspection checklists

- **Checklist of common errors** should be used to drive the inspection.
- Error checklists are programming language dependent and reflect the characteristic errors that are likely to arise in the language.
- In general, the 'weaker' the type checking, the larger the checklist.
- Examples: Initialisation, Constant naming, loop termination, array bounds, etc.

In general, previous projects' "lessons learnt" or final reports would be worth reading.

An inspection checklist (a)

Fault class	Inspection check
Data faults	• Are all program variables initialized before their values are used? • Have all constants been named? • Should the upper bound of arrays be equal to the size of the array or Size -1? • If character strings are used, is a delimiter explicitly assigned? • Is there any possibility of buffer overflow?
Control faults	• For each conditional statement, is the condition correct? • Is each loop certain to terminate? • Are compound statements correctly bracketed? • In case statements, are all possible cases accounted for? • If a break is required after each case in case statements, has it been included?
Input/output faults	• Are all input variables used? • Are all output variables assigned a value before they are output? • Can unexpected inputs cause corruption?

An inspection checklist (b)

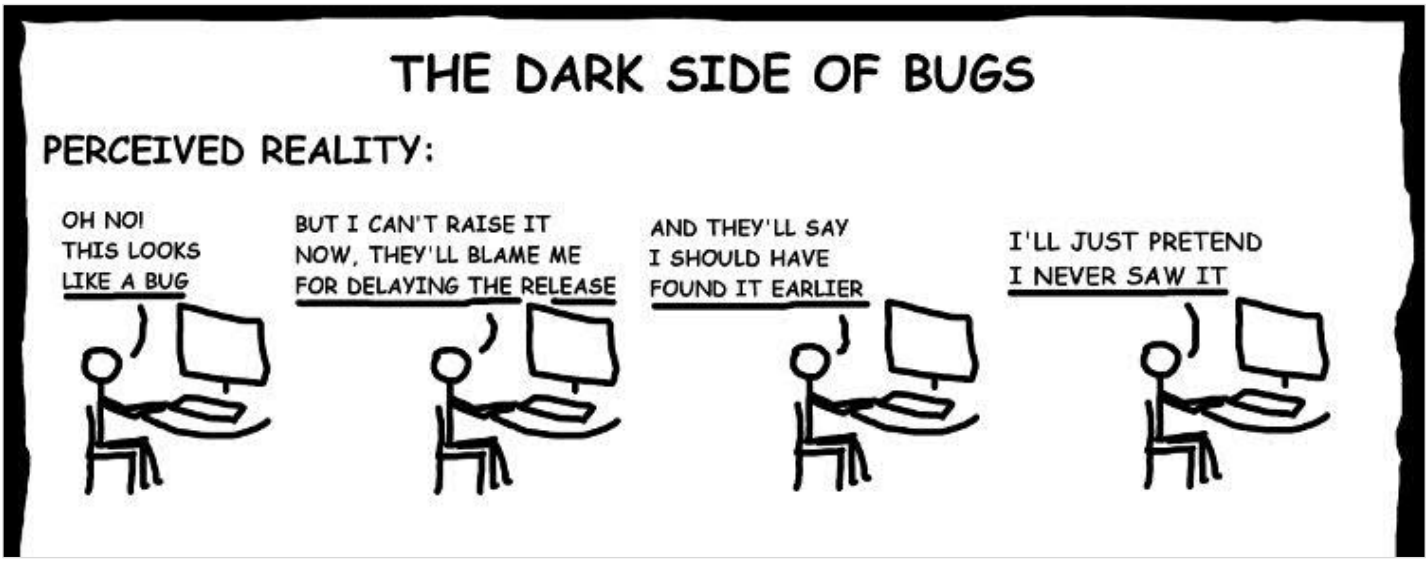
Fault class	Inspection check
Interface faults	• Do all function and method calls have the correct number of parameters? • Do formal and actual parameter types match? • Are the parameters in the right order? • If components access shared memory, do they have the same model of the shared memory structure?
Storage management faults	• If a linked structure is modified, have all links been correctly reassigned? • If dynamic storage is used, has space been allocated correctly? • Is space explicitly deallocated after it is no longer required?
Exception management faults	• Have all possible error conditions been taken into account?

Testing

https://www.tutorialspoint.com/difference-between-testing-and-debugging

Following are the important differences between Testing and Debugging.

Sr. No.	Key	Testing	Debugging
1	Definition	Technically Testing is a process to check if the application is working same as it was supposed to do, and not working as it was not supposed to do.	On other hand Debugging is the activity performed by developers to fix the bug found in the system.
2	Objective	Main objective of Testing is to find bugs and errors in an application which get missed during the unit testing by the developer.	On other hand the main objective of Debugging is to find the exact root cause at code level to fix the errors and bugs found during the testing.
3	Perform	As Testing is mainly to find out the errors and bugs is mainly performed by the testers. Also if testing is at developer end known as unit testing then it is performed by the Developer.	While on other hand Debugging is to find the missing or de-faulty code in an application hence major performed by the developers only.
4	Knowledge Required	As Testing covers the functional and behavioural flow of an application so only functional knowledge is required for the tester to perform the testing.	On other hand Debugging is to find the error at code level so technical and code level knowledge is required for the developer to perform debugging.
5	Automation	Testing can be manual or made automated with the help of different tools.	On other hand Debugging can't be get automated it is always be the manual.
6	Level	Testing on basis of level of performing is at different level i.e., unit testing, integration testing, system testing, etc.	On other hand no such level of Debugging is possible.



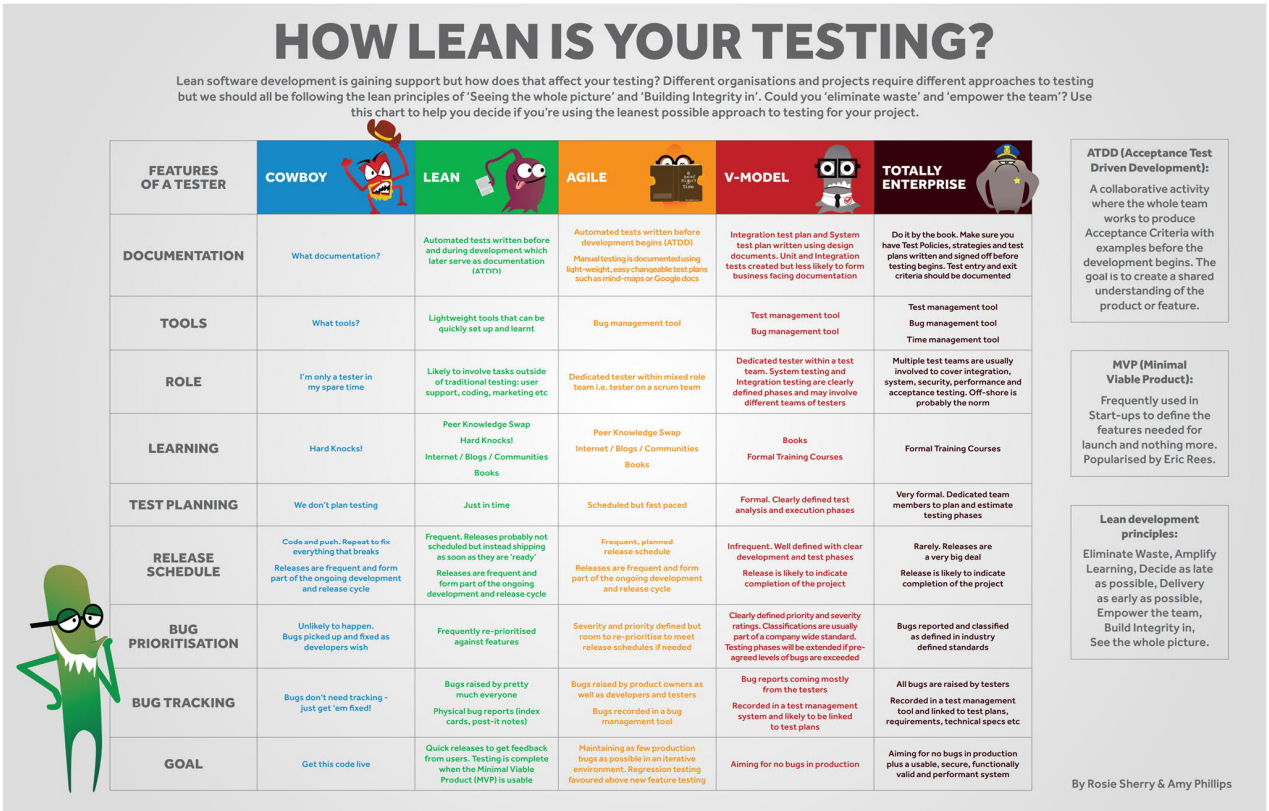
Differences between Testing and Debugging

Last Updated: 27-02-2020

Testing:
Testing is the process of verifying and validating that a software or application is bug free, meets the technical requirements as guided by its design and development and meets the user requirements effectively and efficiently with handling all the exceptional and boundary cases.

Debugging:
Debugging is the process of fixing a bug in the software. It can defined as the identifying, analyzing and removing errors. This activity begins after the software fails to execute properly and concludes by solving the problem and successfully testing the software. It is considered to be an extremely complex and tedious task because errors need to be resolved at all stages of debugging.

[https://www.geeksforgeeks.org/differences-between-testing-and-debugging/]



[https://infotech.report/infographics/how-lean-is-your-testing/8296]

Tampereen yliopisto
Tampere University

Some testing terminology, 1

Alpha testing = last testing round inside company, before beta testing.

Beta testing = trusted partners (or users) test full ”ready” software, before shipment/publishing. Sometimes called pre-release testing.

Monkey testing = ”monkey hits keyboard” or ”cat walks on keyboard”, many surprising errors may be found, is the software stable, however tests are not easily repeatable.

Gorilla testing, depending of the speaker, means whether aggressive nasty testing trying to break the software, or repeating stubbordingly same tests (usually unit or integration).

Tampereen yliopisto
Tampere University

Itasca has four main software development stages: [https://www.itascainternational.com/software/faqs/]

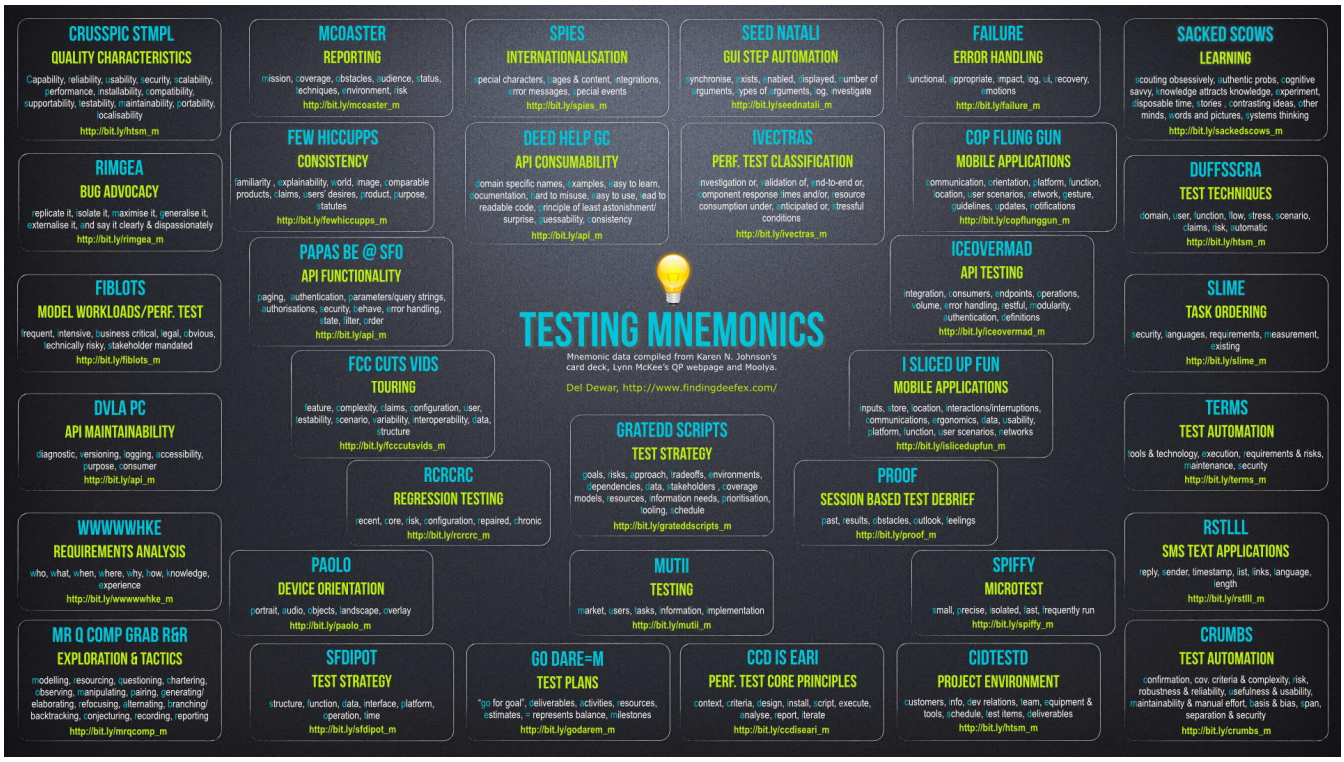
Alpha: Product is actively in development after the previous software release and being tested internally by Itasca consultants.

Beta: Product is feature-complete and being tested internally by Itasca consultants and select clients.

Pre-release: At the discretion of the software Product Manager, software installation and demo files will be publicly available on our website. At a minimum, the Command Reference and FISH Reference manuals will be available. Public information about "What's New" will be distributed and the final date for pre-purchasing the product at a discount will be announced. Clients who have pre-purchased this software can download and use this version.

Release: Pre-purchase discounting ends. The final product is released to the general public and new licenses are distributed to everyone who pre-purchased the software.

Just another… tester's wall table



Types of user testing

✧ Alpha testing

- Users of the software work with the development team to test the software **at the developer's site**.

✧ Beta testing

- A release of the software is **made available to users** to allow them to experiment and to raise problems that they discover with the system developers.

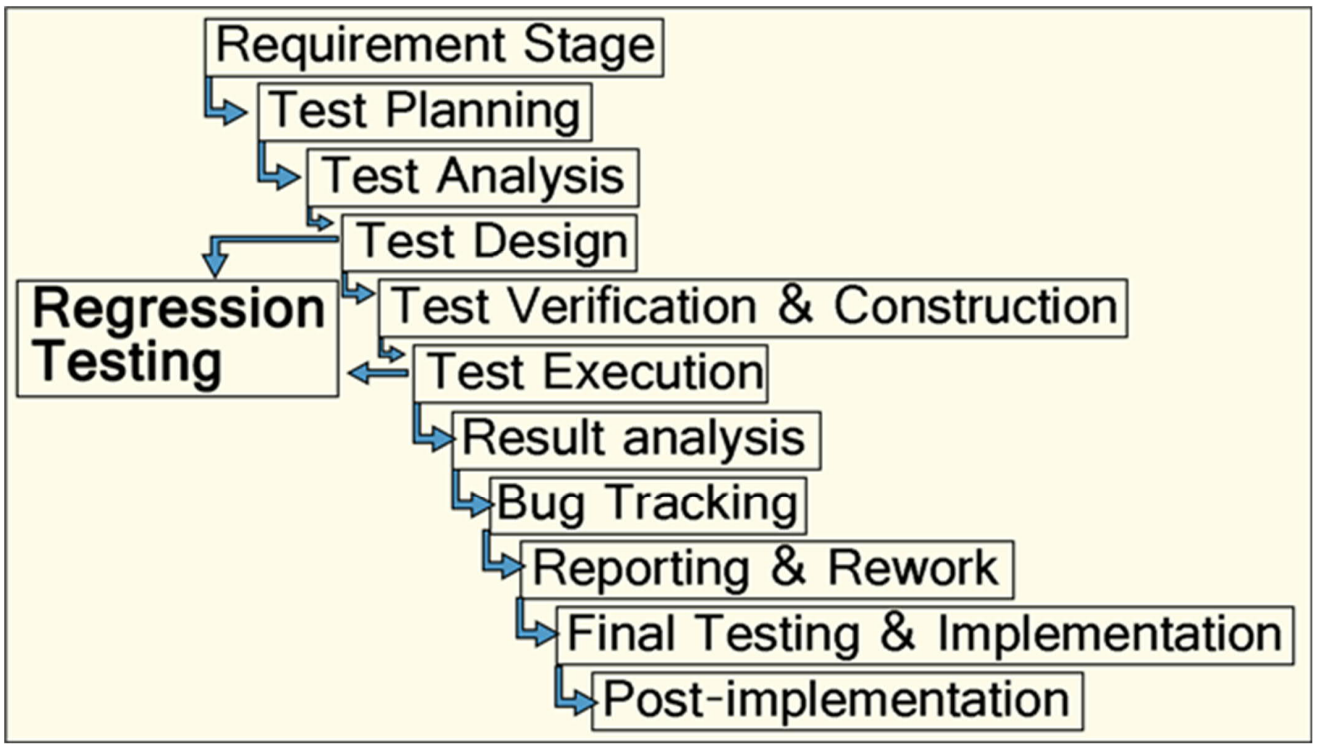
✧ Acceptance testing

- **Customers test a system** to decide whether or not it is ready to be accepted from the system developers and deployed in the customer environment. Primarily for custom systems.

Regression testing

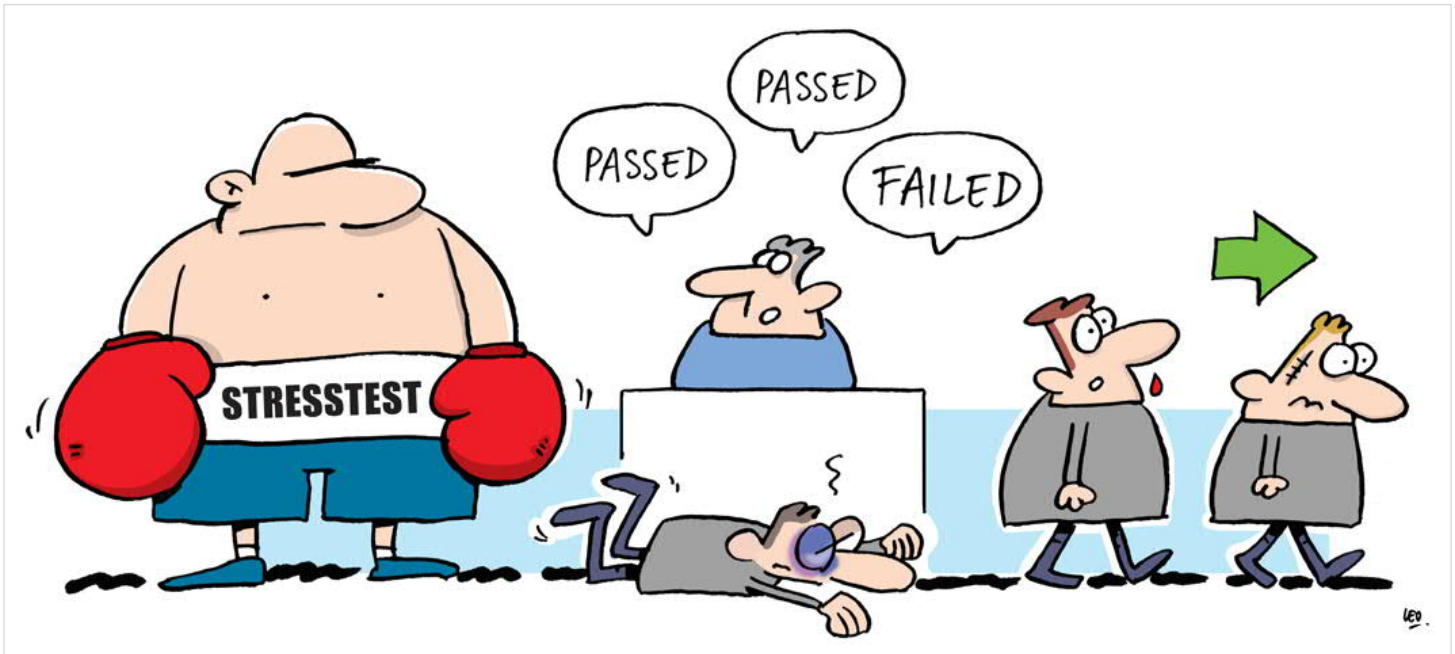
- Regression testing is testing the system **to check that changes have not 'broken' previously working code**.
- In a manual testing process, regression testing is expensive but, with automated testing, it is simple and straightforward. All tests are rerun every time a change is made to the program.
- Tests must run 'successfully' before the change is committed.

To check there have not been any "side-effects" after modifications.



[techspirited.com/software-testing-life-cycle]

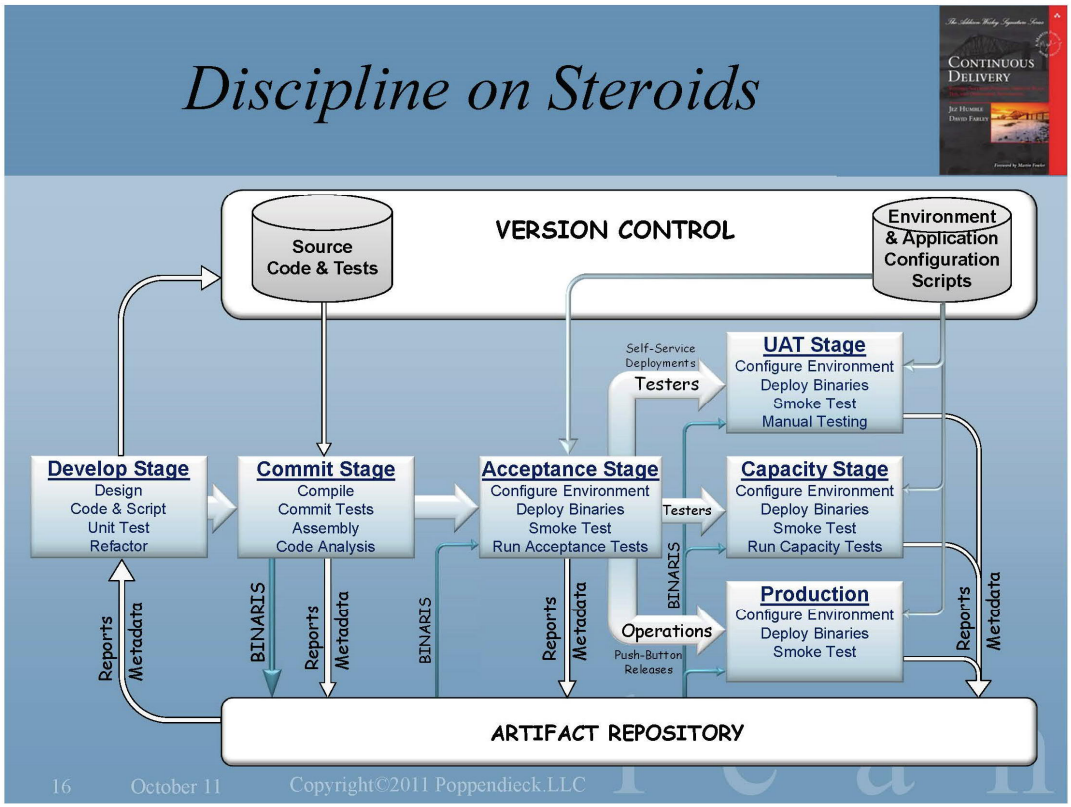
There are many kind of tests and testers...

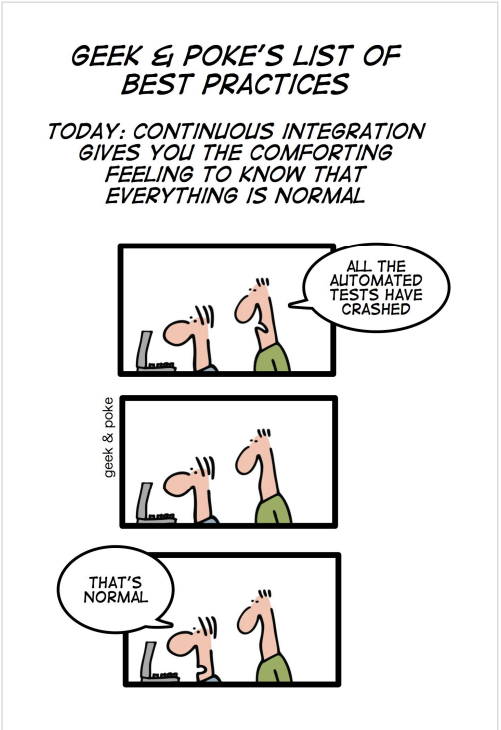


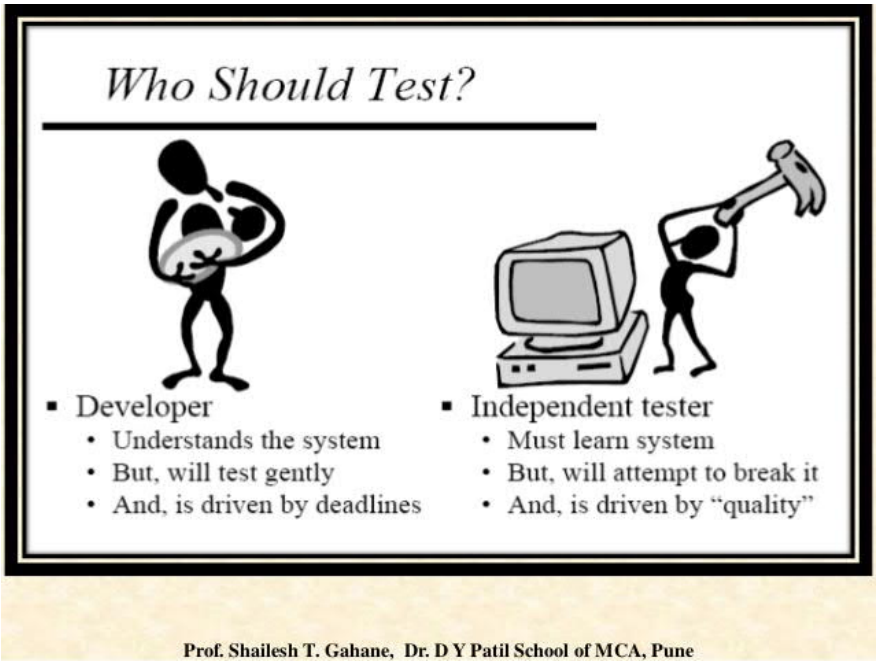
CI = continuous integration

CD = continuous delivery

UAT = user acceptance testing







An official website of the United States government Info

NIST

Search NIST Menu

NEWS

NIST Tool Enables More Comprehensive Tests on High-Risk Software

Updated "combinatorial testing" tool could reduce potential errors in safety-critical applications.

April 23, 2019

Share

We entrust our lives to software every time we step aboard a high-tech aircraft or modern car. A long-term research effort guided by two researchers at the National Institute of Standards and Technology (NIST) and their collaborators has developed new tools to make this type of safety-critical software even safer.

Augmenting an existing software toolkit, the research team's new

MEDIA CONTACT

Chad Boutin
charles.boutin@nist.gov
(301) 975-4261

ORGANIZATIONS

Information Technology Laboratory
Computer Security Division
Security Components and Mechanisms

Tampereen yliopisto
Tampere University

2019

The CWE Top 25

Below is a brief listing of the weaknesses in the 2019 CWE Top 25, including the overall score of each.

Rank	ID	Name	Score
[1]	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	75.56
[2]	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	45.69
[3]	CWE-20	Improper Input Validation	43.61
[4]	CWE-200	Information Exposure	32.12
[5]	CWE-125	Out-of-bounds Read	26.53
[6]	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	24.54
[7]	CWE-416	Use After Free	17.94
[8]	CWE-190	Integer Overflow or Wraparound	17.35
[9]	CWE-352	Cross-Site Request Forgery (CSRF)	15.54
[10]	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	14.10
[11]	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	11.47
[12]	CWE-787	Out-of-bounds Write	11.08
[13]	CWE-287	Improper Authentication	10.78
[14]	CWE-476	NULL Pointer Dereference	9.74
[15]	CWE-732	Incorrect Permission Assignment for Critical Resource	6.33
[16]	CWE-434	Unrestricted Upload of File with Dangerous Type	5.50
[17]	CWE-611	Improper Restriction of XML External Entity Reference	5.48
[18]	CWE-94	Improper Control of Generation of Code ('Code Injection')	5.36
[19]	CWE-798	Use of Hard-coded Credentials	5.12
[20]	CWE-400	Uncontrolled Resource Consumption	5.04
[21]	CWE-772	Missing Release of Resource after Effective Lifetime	5.04
[22]	CWE-426	Untrusted Search Path	4.40

[https://cwe.mitre.org/top25/archive/2019/2019_cwe_top25.html]

Tampereen yliopisto
Tampere University

2020

The CWE Top 25

Below is a brief listing of the weaknesses in the 2020 CWE Top 25, including the overall score of each.

Rank	ID	Name	Score
[1]	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	46.82
[2]	CWE-787	Out-of-bounds Write	46.17
[3]	CWE-20	Improper Input Validation	33.47
[4]	CWE-125	Out-of-bounds Read	26.50
[5]	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	23.73
[6]	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	20.69
[7]	CWE-200	Exposure of Sensitive Information to an Unauthorized Actor	19.16
[8]	CWE-416	Use After Free	18.87
[9]	CWE-352	Cross-Site Request Forgery (CSRF)	17.29
[10]	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	16.44
[11]	CWE-190	Integer Overflow or Wraparound	15.81
[12]	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	13.67
[13]	CWE-476	NULL Pointer Dereference	8.35
[14]	CWE-287	Improper Authentication	8.17
[15]	CWE-434	Unrestricted Upload of File with Dangerous Type	7.38
[16]	CWE-732	Incorrect Permission Assignment for Critical Resource	6.95
[17]	CWE-94	Improper Control of Generation of Code ('Code Injection')	6.53
[18]	CWE-522	Insufficiently Protected Credentials	5.49
[19]	CWE-611	Improper Restriction of XML External Entity Reference	5.33
[20]	CWE-798	Use of Hard-coded Credentials	5.19
[21]	CWE-502	Deserialization of Untrusted Data	4.93
[22]	CWE-269	Improper Privilege Management	4.87

[https://cwe.mitre.org/top25/archive/2020/2020_cwe_top25.html]

Tampereen yliopisto
Tampere University

Web testing

Web testing is a software testing practice to test websites or web applications for potential bugs. It's a complete testing of web-based applications before making live.

A web-based system needs to be checked completely from end-to-end before it goes live for end users.

By performing website testing, an organization can make sure that the web-based system is functioning properly and can be accepted by real-time users.

The UI design and functionality are the captains of website testing.

Web Testing Checklists

1) Functionality Testing
2) Usability testing
3) Interface testing
4) Compatibility testing
5) Performance testing
6) Security testing.

[https://www.softwaretestinghelp.com/web-application-testing/]

Tampereen yliopisto
Tampere University

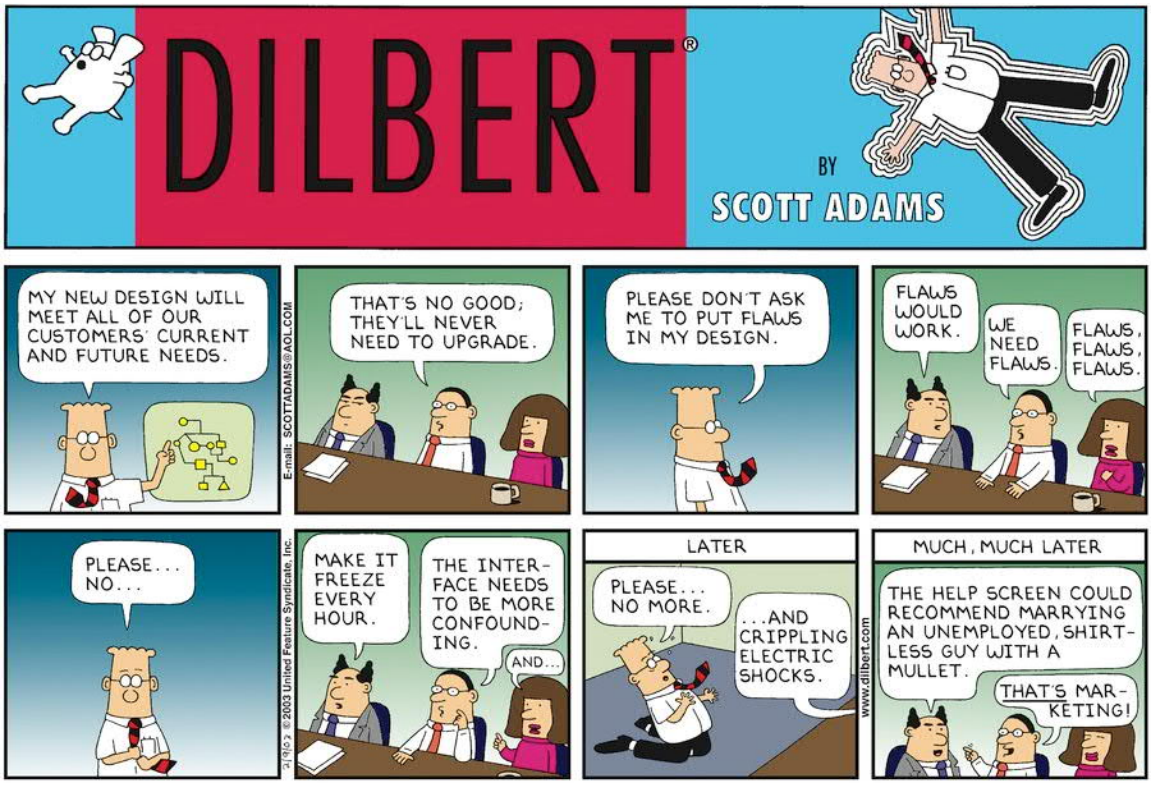
Practical Software Testing – E-Book by http://www.SoftwareTestingHelp.com Version 2.0

Practical Software Testing - eBook

Prepared by Chindam Damodar
Published by http://SoftwareTestingHelp.com

Tampereen yliopisto
Tampere University

[https://work2code.com/software-testing-learning-promises-2020/]



Another "work flow" for later stages of testing...

Pre-Alpha:- Software is a prototype. UI is complete. But not all features are completed. At this stage, software is not published.

Alpha: Software is near its development and is internally tested for bugs/issues

Beta: Software is stable and is released to a limited user base. The goal is to get customer feedback on the product and make changes in software accordingly

Release Candidate (RC): Based on the feedback of Beta Test, you make changes to the software and want to test out the bug fixes. At this stage, you do not want to make radical changes in functionality but just check for bugs. RC is also put out to the public

Release: All works, software is released to the public.

Note: Above is a standard definition of the Testing stages but in order to garner marketing buzz, companies combine stages like "pre-alpha beta", "pre-beta" etc.

[https://www.guru99.com/alpha-beta-testing-demystified.html]

Some testing terminology, 2

Remember also reviews and inspections, before testing.

- black-box testing (FI: mustalaatikkotestaus), tester do not know the internal structure of the program
- gray-box testing (FI: harmaalaatikkotestaus), tester has some information about the internal functionality of the program, e.g. design document is available
- white-box testing = glass box testing (FI: lasilaatikkotestaus, valkolaatikkotestaus), source code is available.

Black-box testing approach

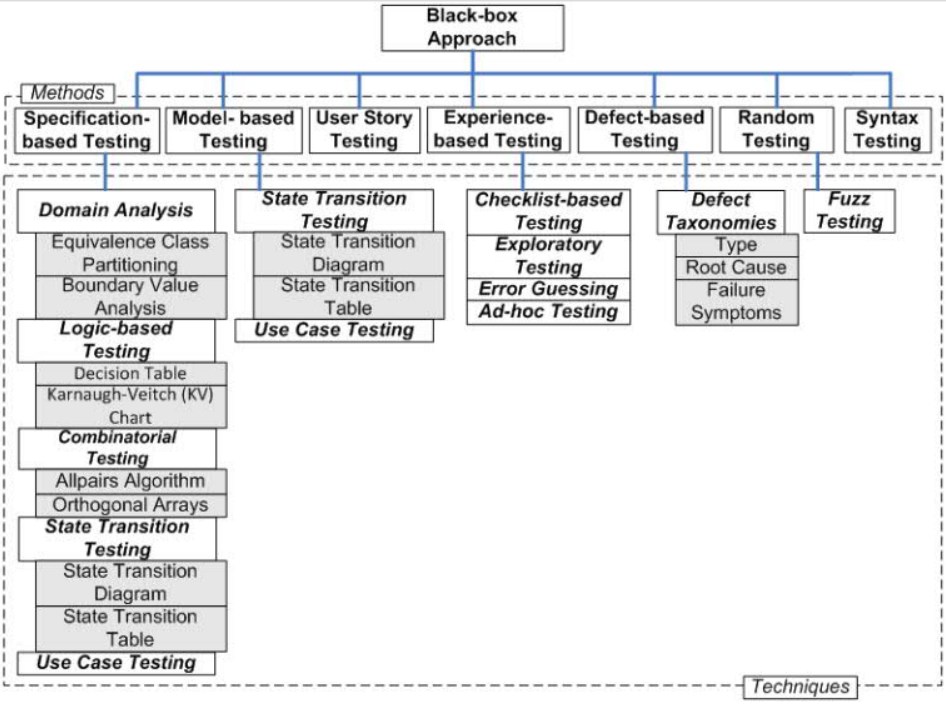


Fig.5. Black-box Approach

[Kulesovs et al., Inventory of Testing Ideas and Structuring of Testing Terms, 2013]

White-box testing approach

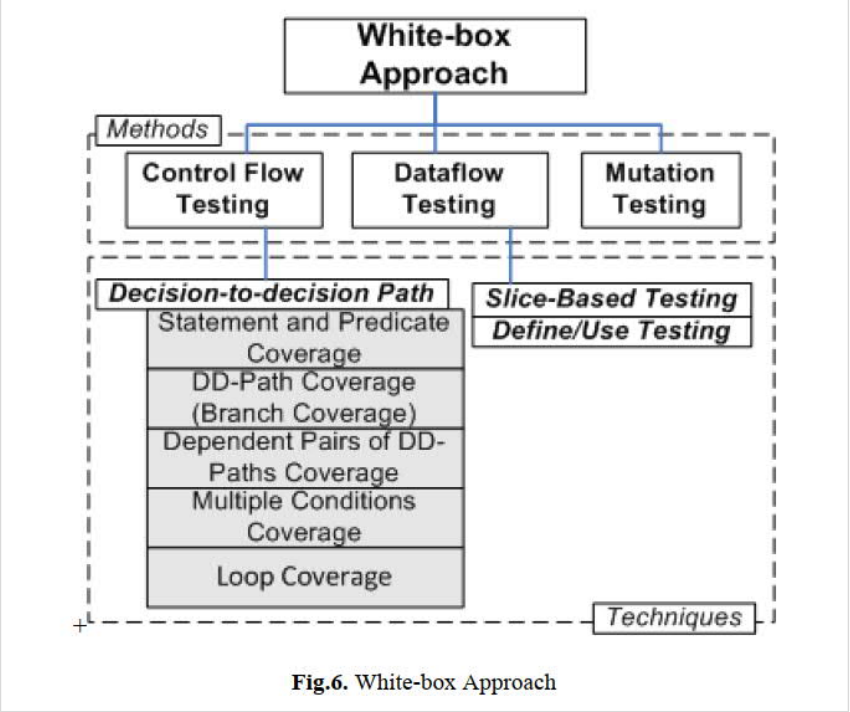
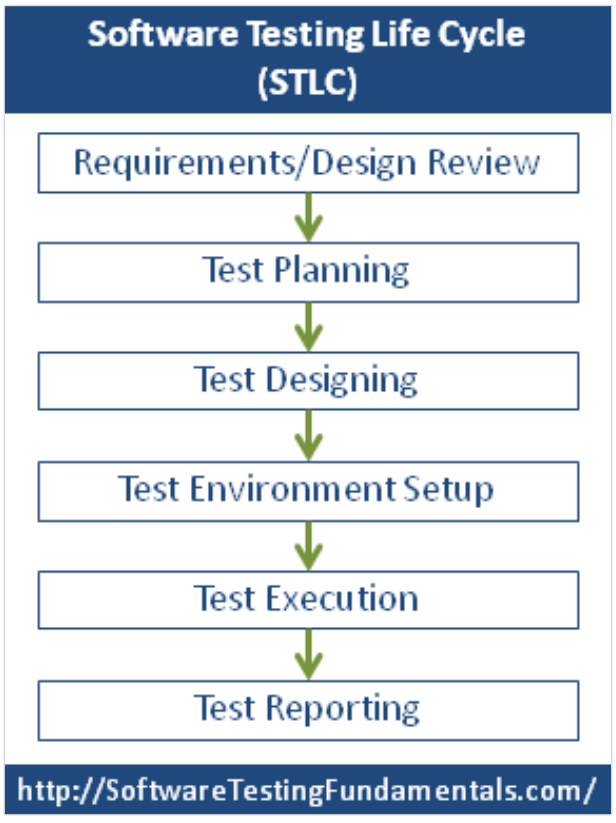


Fig.6. White-box Approach

[Kulesovs et al., Inventory of Testing Ideas and Structuring of Testing Terms, 2013]

You have to PLAN your test, and also have some LOG or REPORT to prove you have tested something.
There must be some evidence of testing.

Just to say ”we did exploratory testing two hours, and did not found any bugs” is NOT enough.
Think yourself, if you were the customer.



[https://pbs.twimg.com/media/CSqxyLGVAAAtUT0.png]

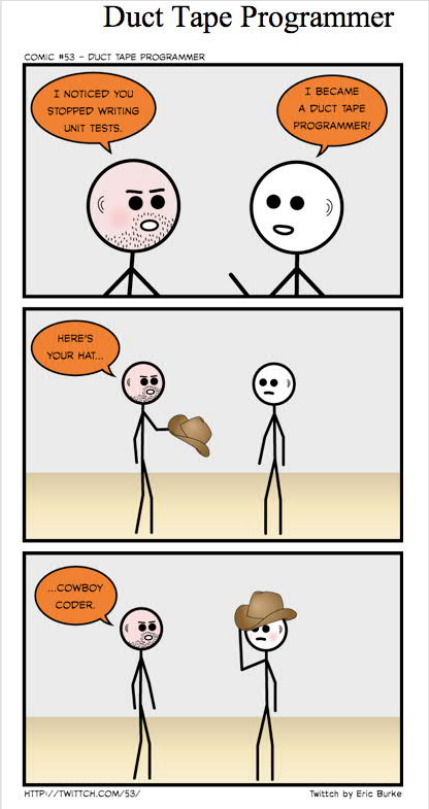
Testing dichotomies

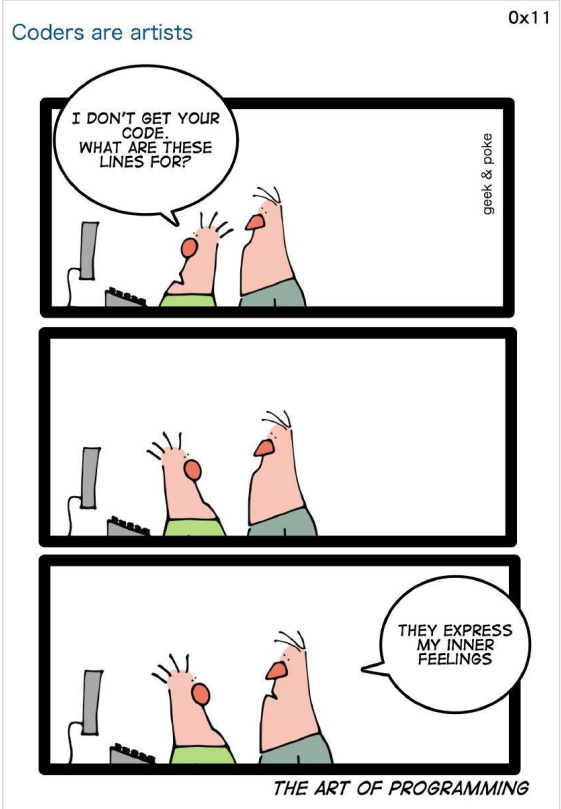
- testing vs. debugging
- black-box testing vs. white-box testing
- functional testing vs. non-functional testing
- manual testing vs. automated testing
- scripted testing vs. exploratory testing
- verification vs. validation
- positive testing vs. negative testing
- testing of design vs. testing of implementation
- static testing vs. dynamic testing
- hierarchical vs. big bang
- traditional testing vs. agile testing.

Sometimes the best results may be achieved by using both methods/types.

Those testing methods or styles are not exclusive.

[Kulesovs et al., Inventory of Testing Ideas and Structuring of Testing Terms, 2013]





The QA Handbook (QAH) is a non-normative handbook about the process and operational aspects of certain quality assurance practices of W3C's Working Groups, with particular focus on testability and test topics. It is intended for Working Group chairs and team contacts. It aims to help them to avoid known pitfalls and benefit from experiences gathered from the W3C Working Groups themselves. It provides techniques, tools, and templates that should facilitate and accelerate their work.

W3C Working Group Note

W3C

The QA Handbook

W3C Working Group Note 6 September 2005

This version:
http://www.w3.org/TR/2005/NOTE-qa-handbook-20050906/
Latest version:
http://www.w3.org/TR/qa-handbook/
Previous version:
http://www.w3.org/TR/2004/NOTE-qa-handbook-20041122/
Editor:
Lofton Henderson
Contributors:
See Acknowledgments.

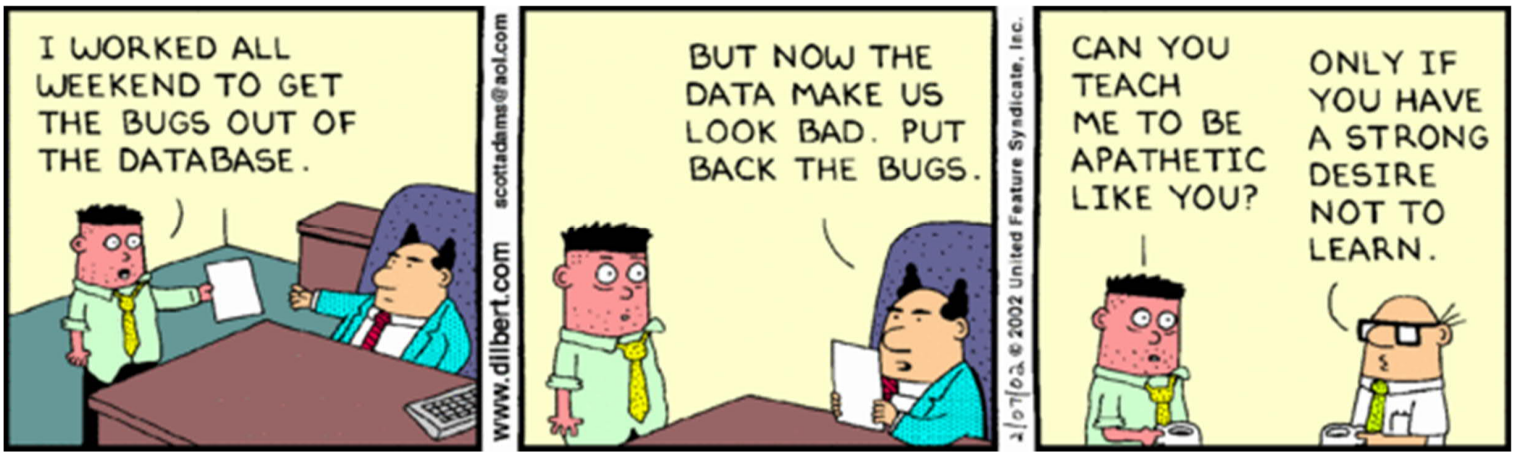
While formulating a code, the following should be kept in mind

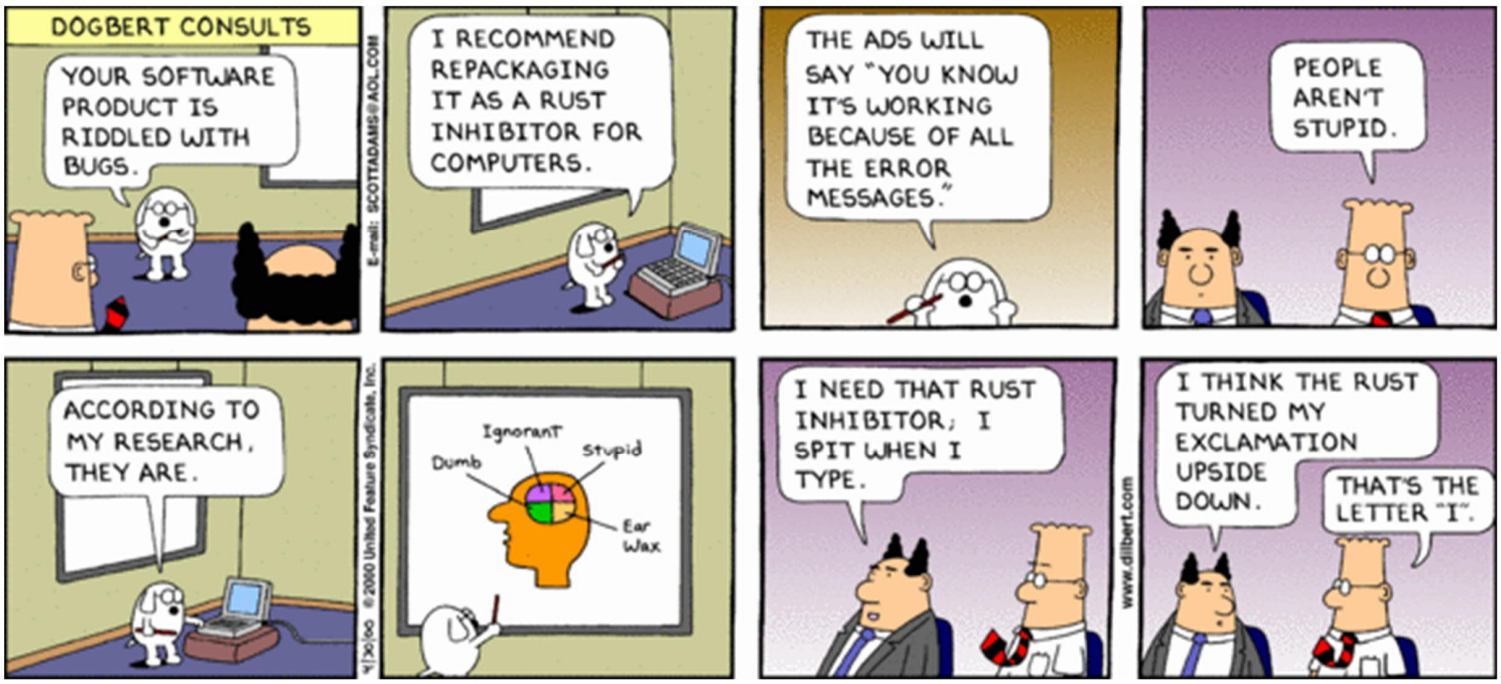
The code should be easy to be read, for this:

- Try to define different sections of the code by segmenting blocks of code into a paragraph
- Make use of indentation for indicating the start and end of the control structures along with a clear specification of where the code is between them
- There should be consistency in the naming convention of the variables throughout the code. Also, the data should be described that is there in the code
- Name the functions according to what they perform
- The code should be such that one should be able to understand it even after returning to it after some time gap, without that person having to look at every line of it
- Follow a specific method for commenting on the work
- The language functions that are complex or the structure that is difficult to be comprehended should be avoided.

There are many advantages to the following coding standards while coding the software.

[https://www.multidots.com/importance-of-code-quality-and-coding-standard-in-software-development/]

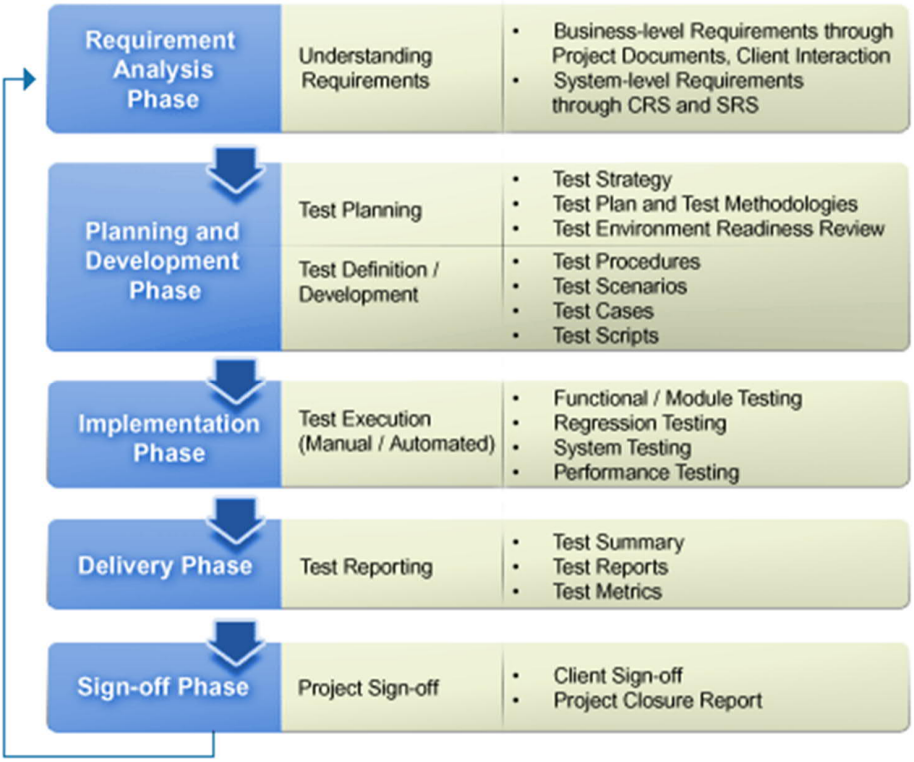




Tampereen yliopisto
Tampere University

Tampereen yliopisto
Tampere University

Testing in SDLC phases



[www.360logica.com/test-management/test-models-planning/]

Tampereen yliopisto
Tampere University

Baselines

A Concept of Operations Baseline

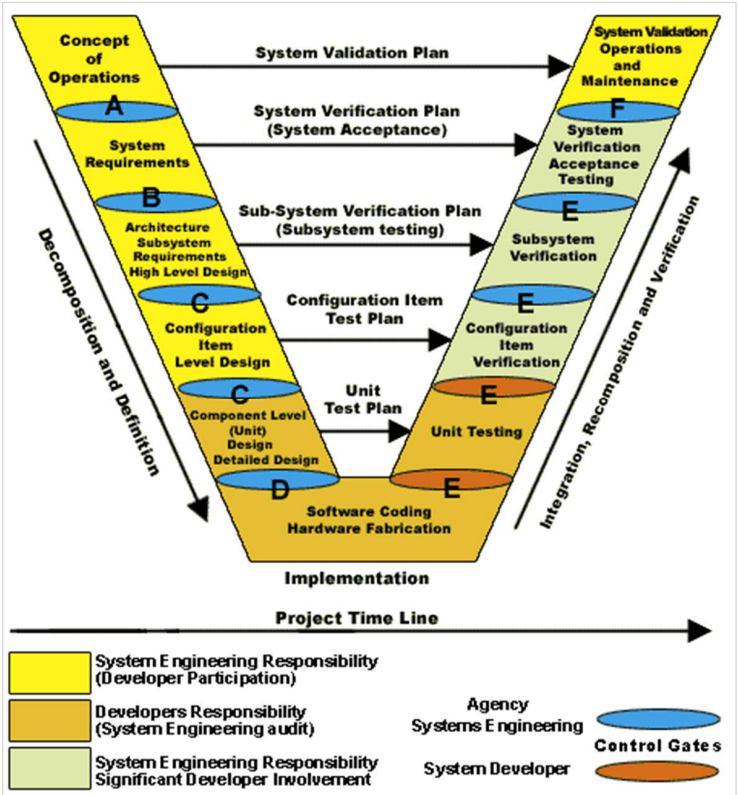
B System Baseline

C Subsystem Baseline

D Development Baseline

E Product Baseline

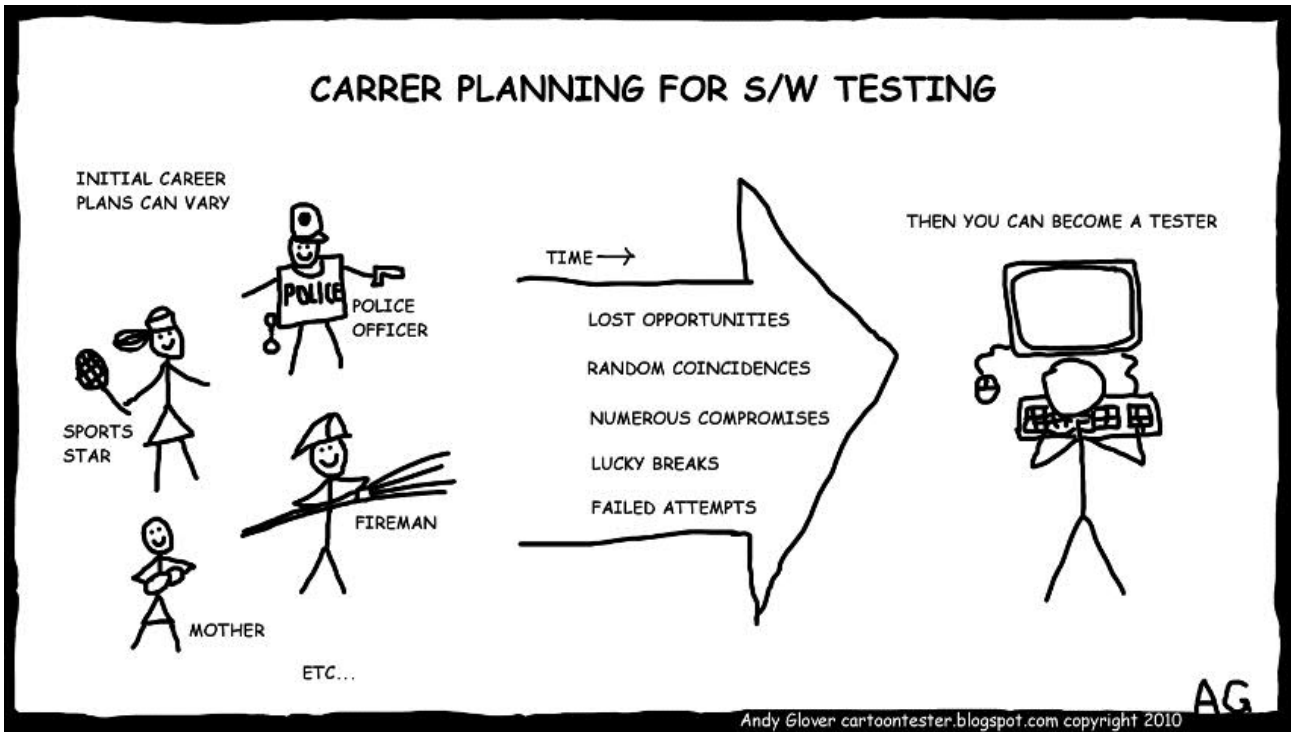
F Operational Baseline

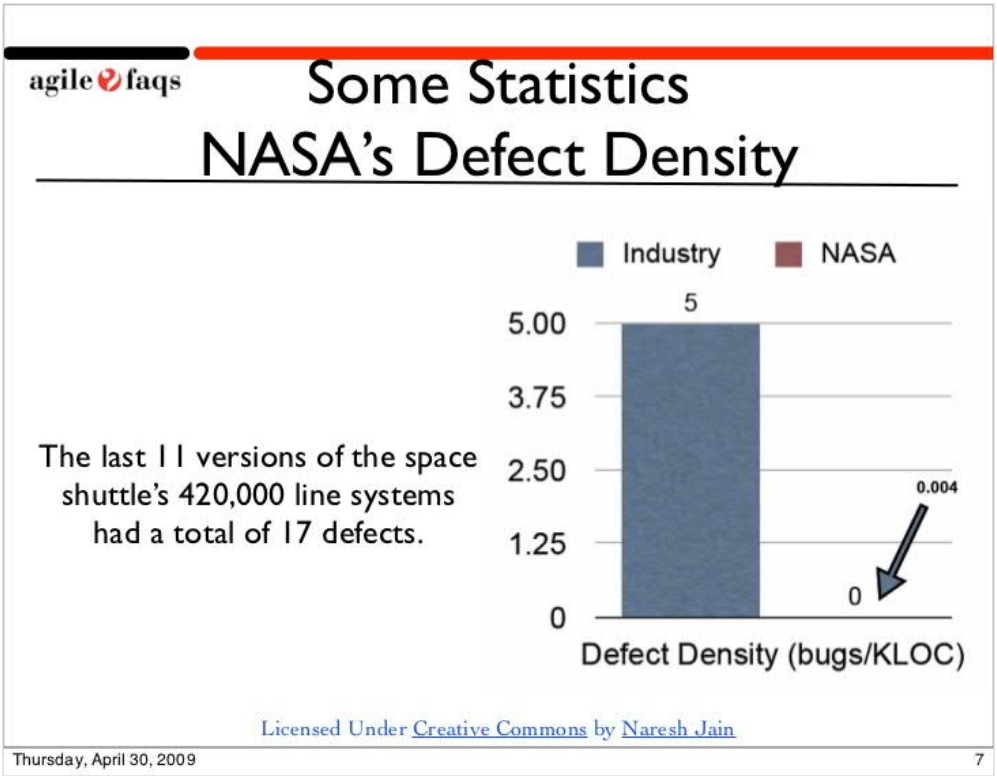


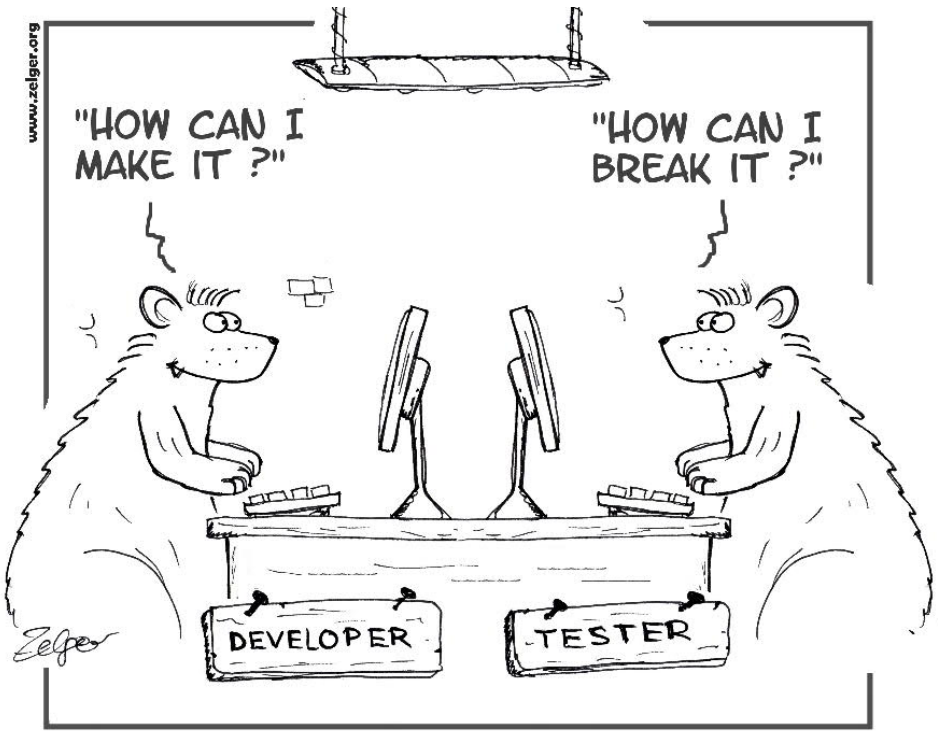
[https://ops.fhwa.dot.gov/freewaymgmt/publications/cm/handbook/]

Pre-launch testing for mobile games

[https://medium.com/googleplaydev/test-pre-launch-96d0ef3c4d51]


CARRER PLANNING FOR S/W TESTING
INITIAL CAREER PLANS CAN VARY
POLICE
POLICE OFFICER
SPORTS STAR
FIREMAN
MOTHER
ETC...
TIME
LOST OPPORTUNITIES
RANDOM COINCIDENCES
NUMEROUS COMPROMISES
LUCKY BREAKS
FAILED ATTEMPTS
THEN YOU CAN BECOME A TESTER
AG
Andy Glover cartoontester.blogspot.com copyright 2010


agile faqs
Some Statistics
NASA's Defect Density
The last 11 versions of the space shuttle's 420,000 line systems had a total of 17 defects.
Industry
NASA
5
5.00
3.75
2.50
1.25
0
0.004
0
Defect Density (bugs/KLOC)
Licensed Under Creative Commons by Naresh Jain
Thursday, April 30, 2009
7



They weren't so much different, but they had different goals

https://softwaretesters.net/

Error density metrics

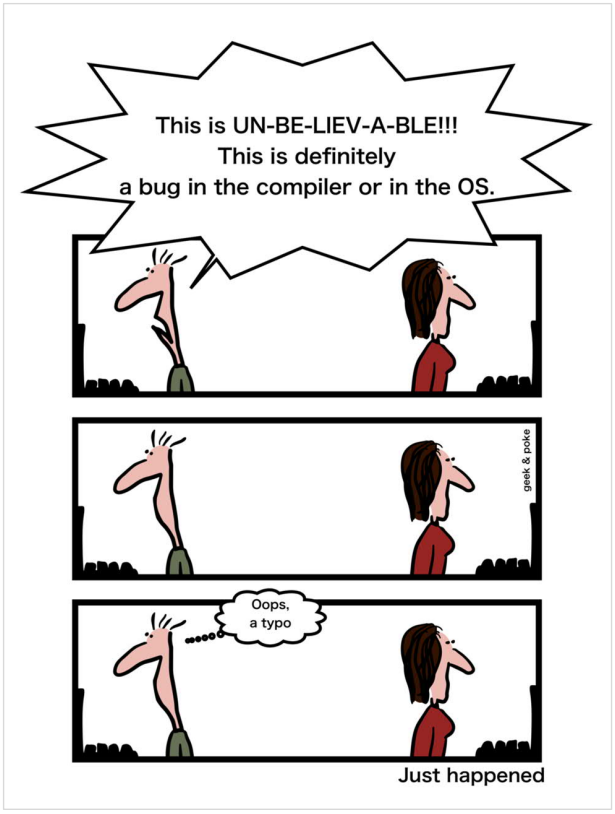
Code	Name	Calculation formula
CED	Code Error Density	$CED = \frac{NCE}{KLOC}$
DED	Development Error Density	$DED = \frac{NDE}{KLOC}$
WCED	Weighted Code Error Density	$WCDE = \frac{WCE}{KLOC}$
WDED	Weighted Development Error Density	$WDED = \frac{WDE}{KLOC}$
WCEF	Weighted Code Errors per Function Point	$WCEF = \frac{WCE}{NFP}$
WDEF	Weighted Development Errors per Function Point	$WDEF = \frac{WDE}{NFP}$

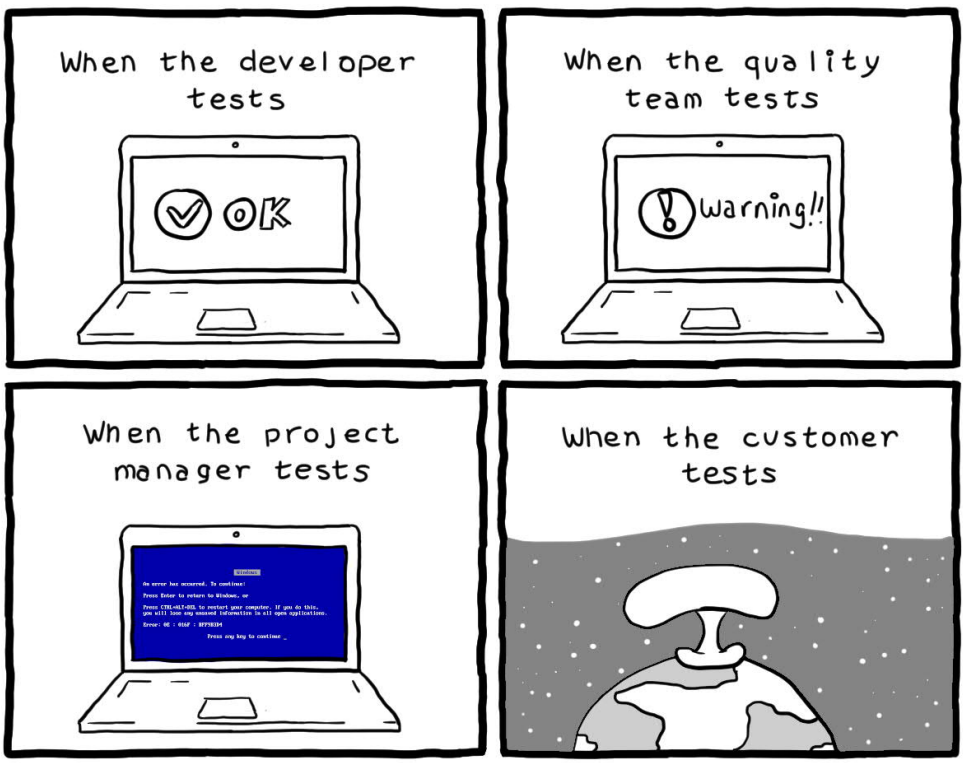
NCE = The number of code errors detected by code inspections and testing.
NDE = total number of development (design and code) errors) detected in the development process.
WCE = weighted total code errors detected by code inspections and testing.
WDE = total weighted development (design and code) errors detected in development process.

Average defect density by software type

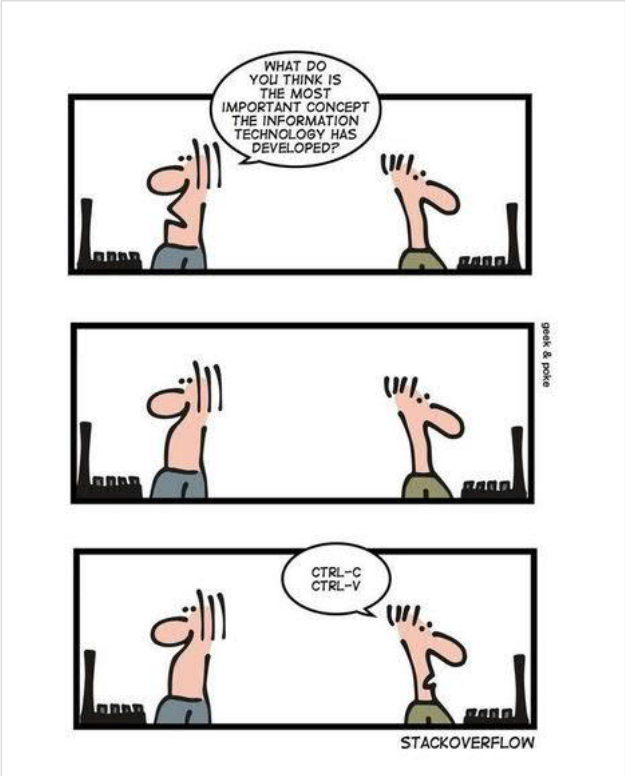
Software application type	Average fielded defect density	Average testing defect density	Ratio of test to field
Wireless capabilities	0.165	3.092	18.7
Biometrics	0.400	1.290	3.2
Domain knowledge can be acquired via public domain in short period of time	0.068	n/a	n/a
Client server	0.108	0.434	4.0
Real time	0.476	2.172	4.6
Multi-tasking	0.449	2.104	4.7
DB interfaces	0.456	1.459	3.2
Mathematically intensive	0.430	1.513	3.5
Web based	0.008	0.091	11.1
Target HW is new or evolving	0.642	2.588	4.0
Application process evolving	0.378	0.278	0.7

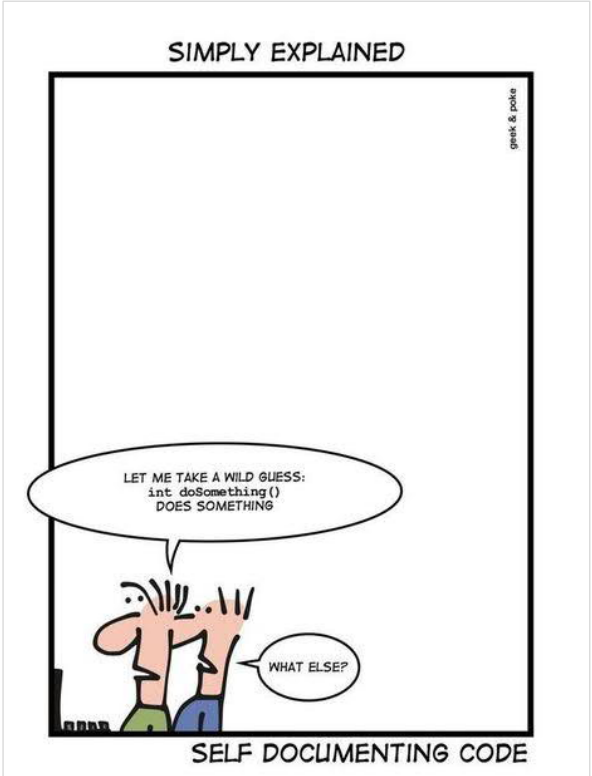
[source: ?????]

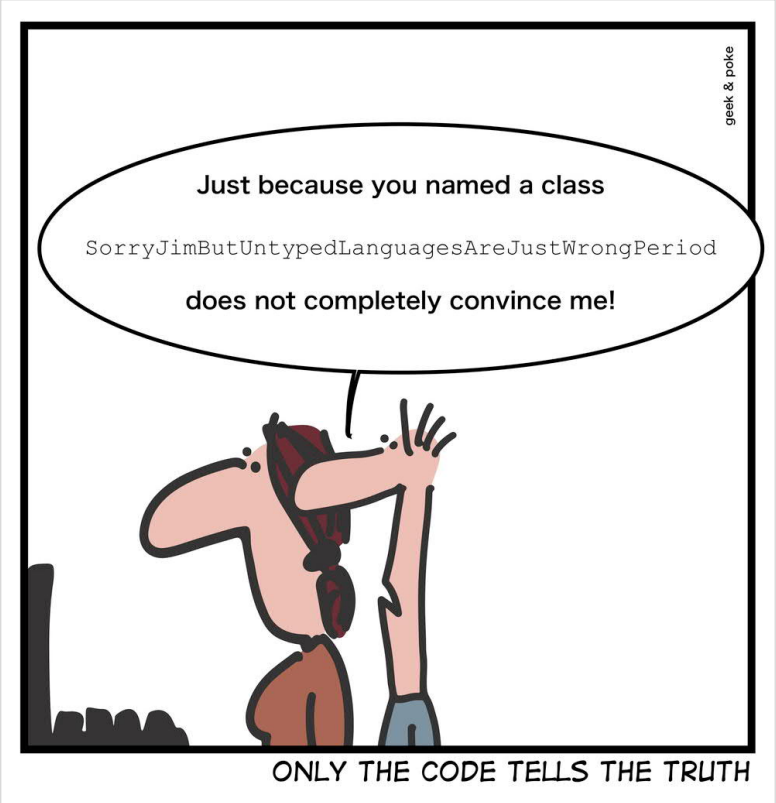

This is UN-BE-LIEV-A-BLE!!!
This is definitely
a bug in the compiler or in the OS.
geek & poke
Oops,
a typo
Just happened


When the developer tests
OK
When the quality team tests
Warning!!
When the project manager tests
When the customer tests

Code style guide
Coding conventions







CI = continuous integration

CD = continuous deployment

CI / CD

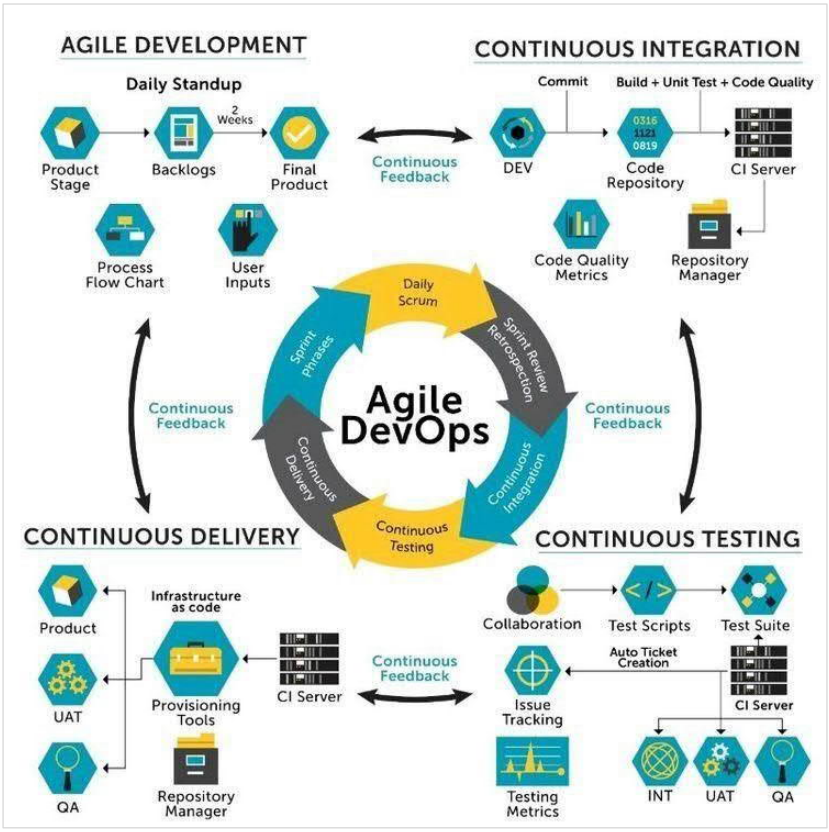
- Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible. Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly.

[Martin Fowler, 2006]

Continuous deployment

- Continuous deployment can be thought of as an extension of continuous integration, aiming at minimizing lead time, the time elapsed between development writing one new line of code and this new code being used by live users, in production.
- To achieve continuous deployment, the team relies on infrastructure that automates and instruments the various steps leading up to deployment, so that after each integration successfully meeting these release criteria, the live application is updated with new code.
- Instrumentation is needed to ensure that any suggestion of lowered quality results in aborting the deployment process, or rolling back the new features, and triggers human intervention.

[Agile Alliance]



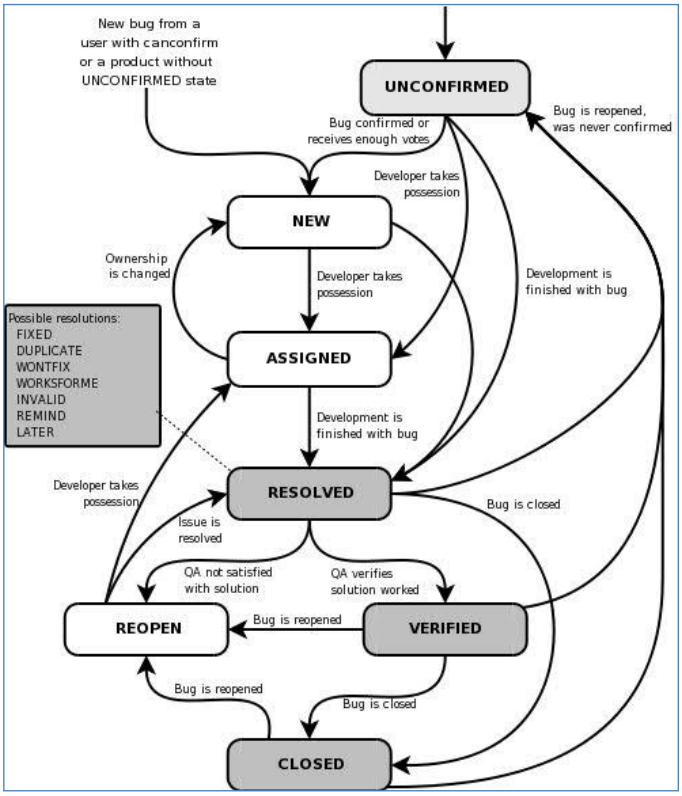
automated testing

[Jukka Ala-Mutka]

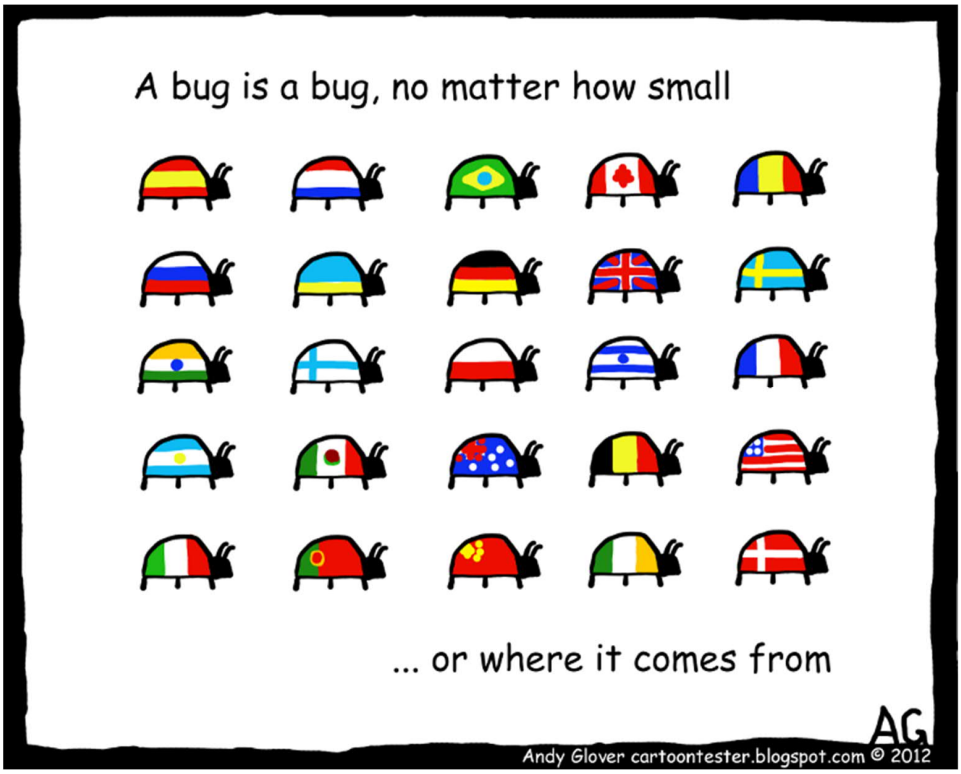
Tampereen yliopisto
Tampere University

Bug report(ing)

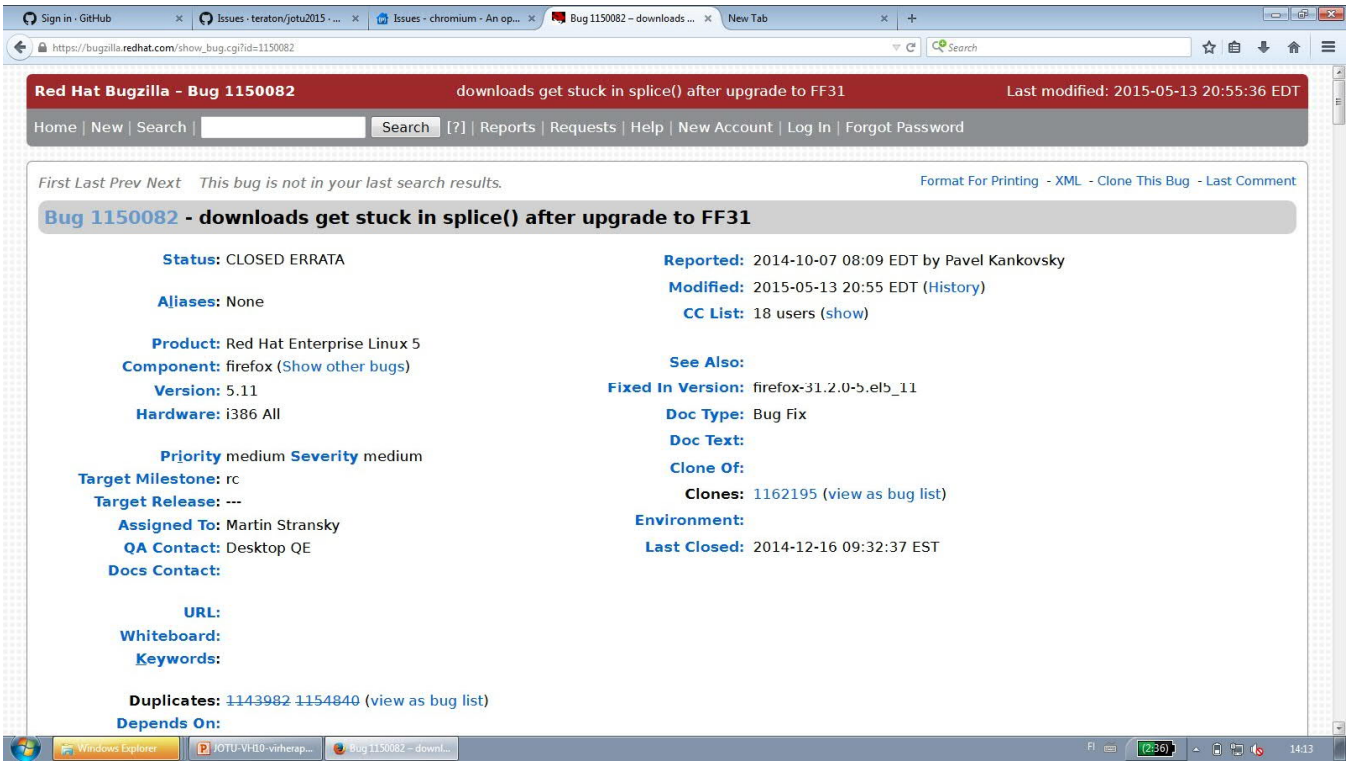
One example of bug tracking process; possible stages and resolutions of a reported bug



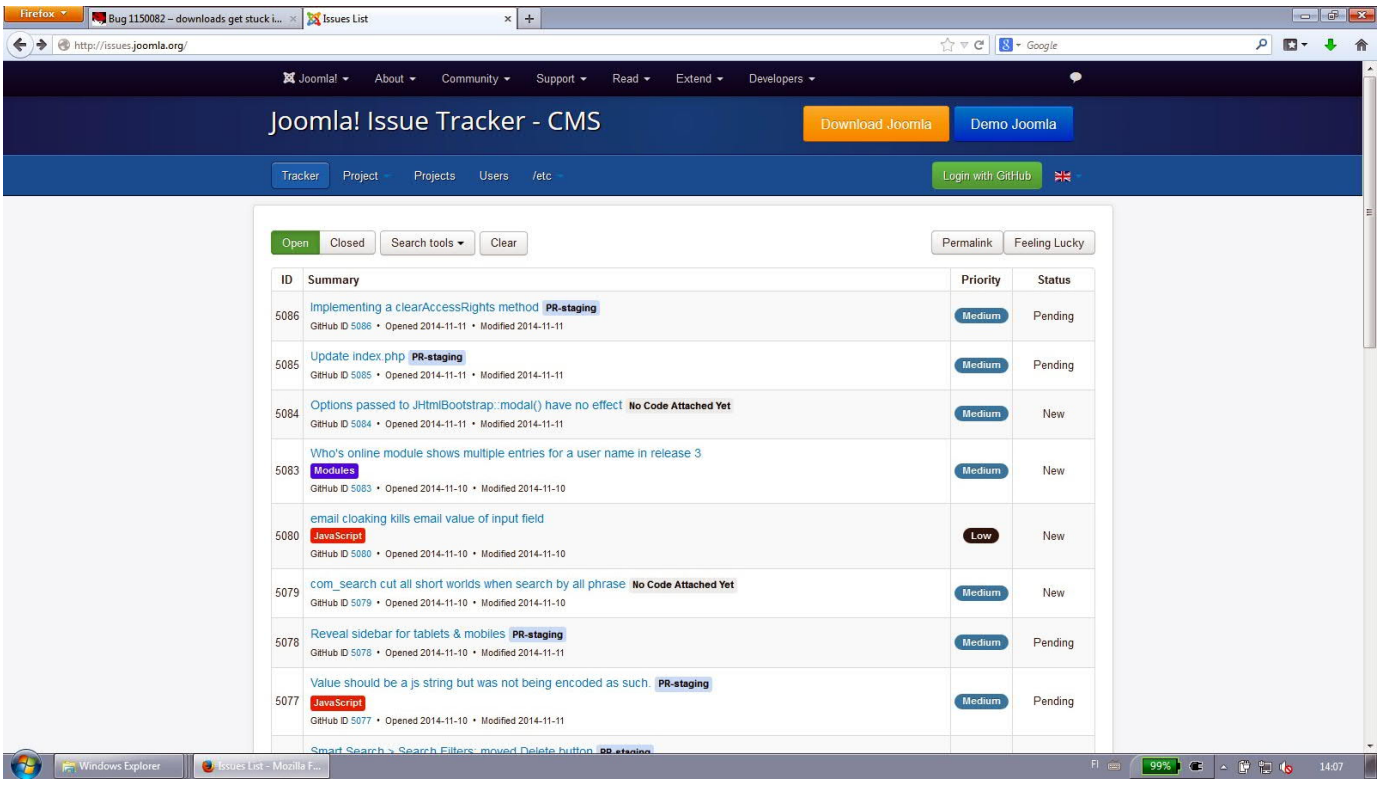
http://www.softwaretestinghelp.com/



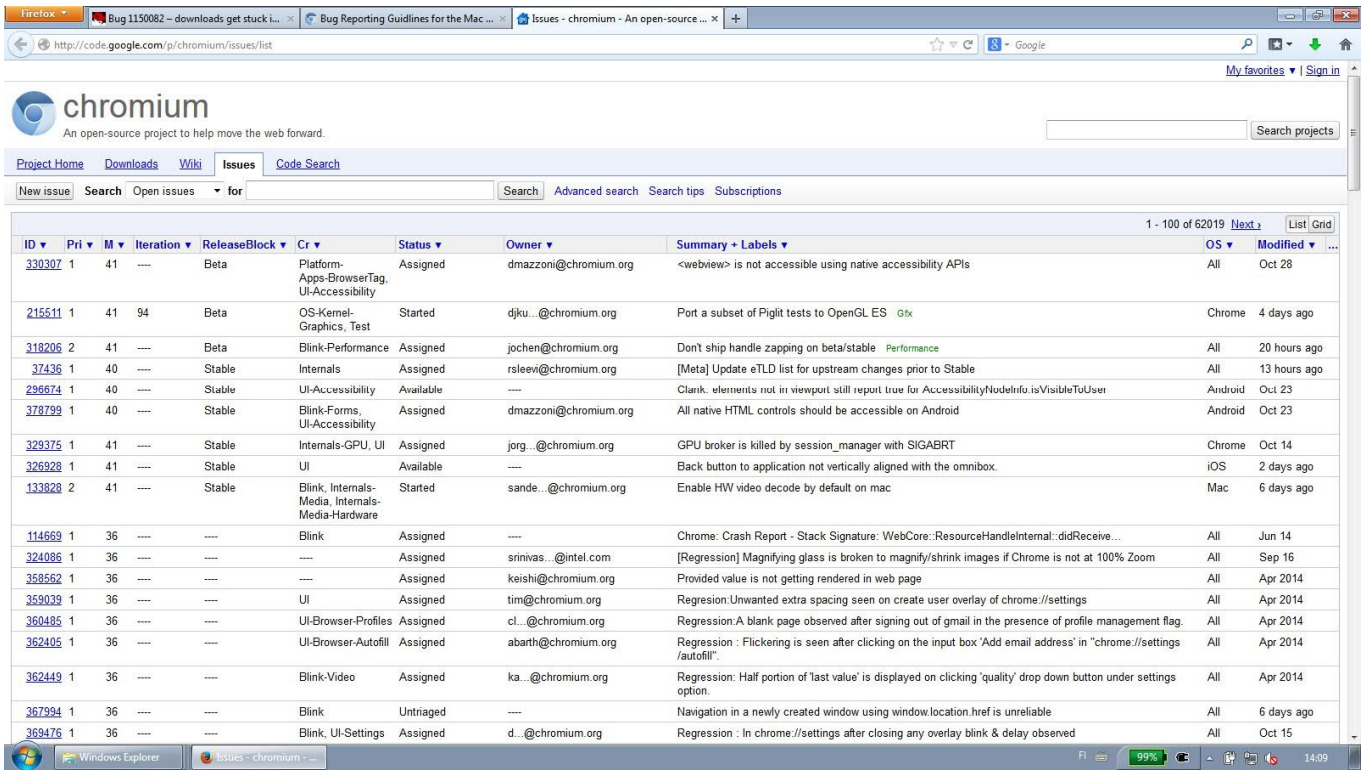
https://bugzilla.redhat.com/show_bug.cgi?id=1150082

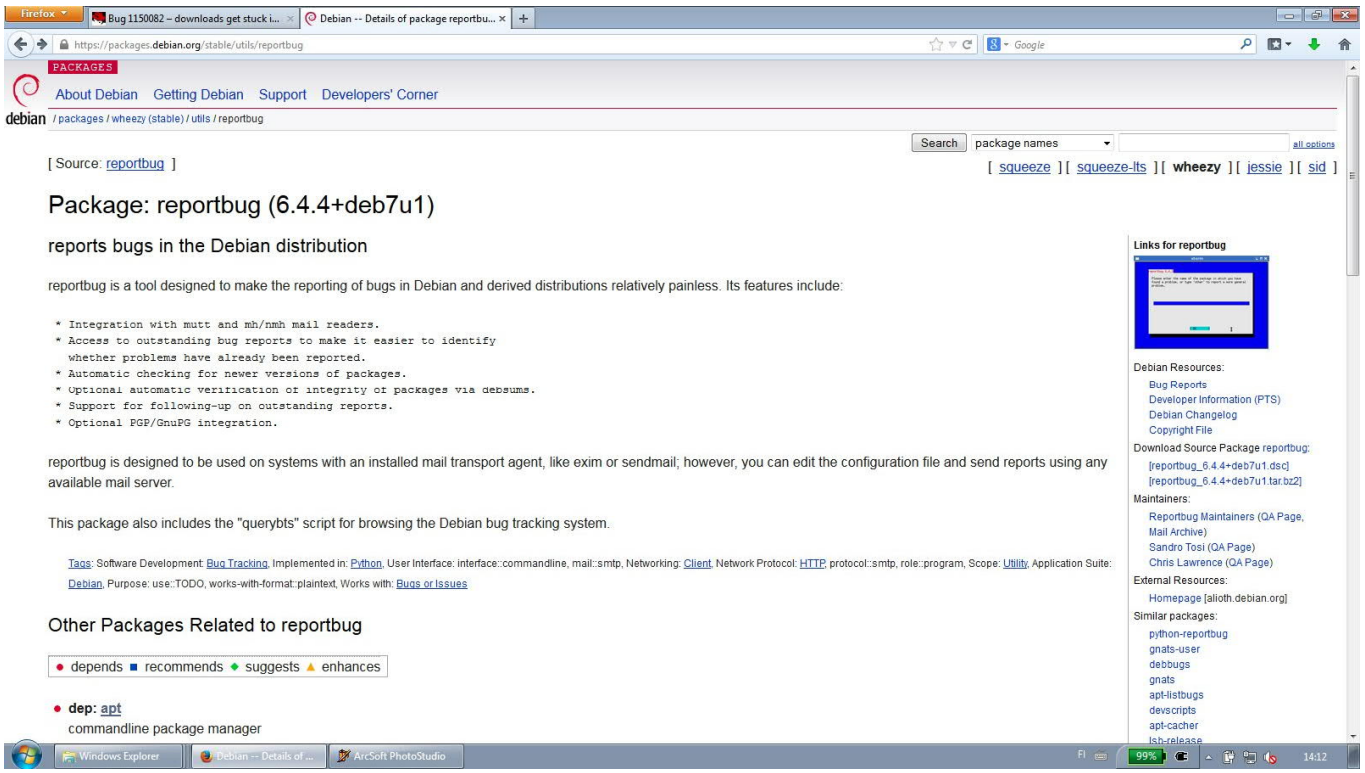


issues.joomla.org/

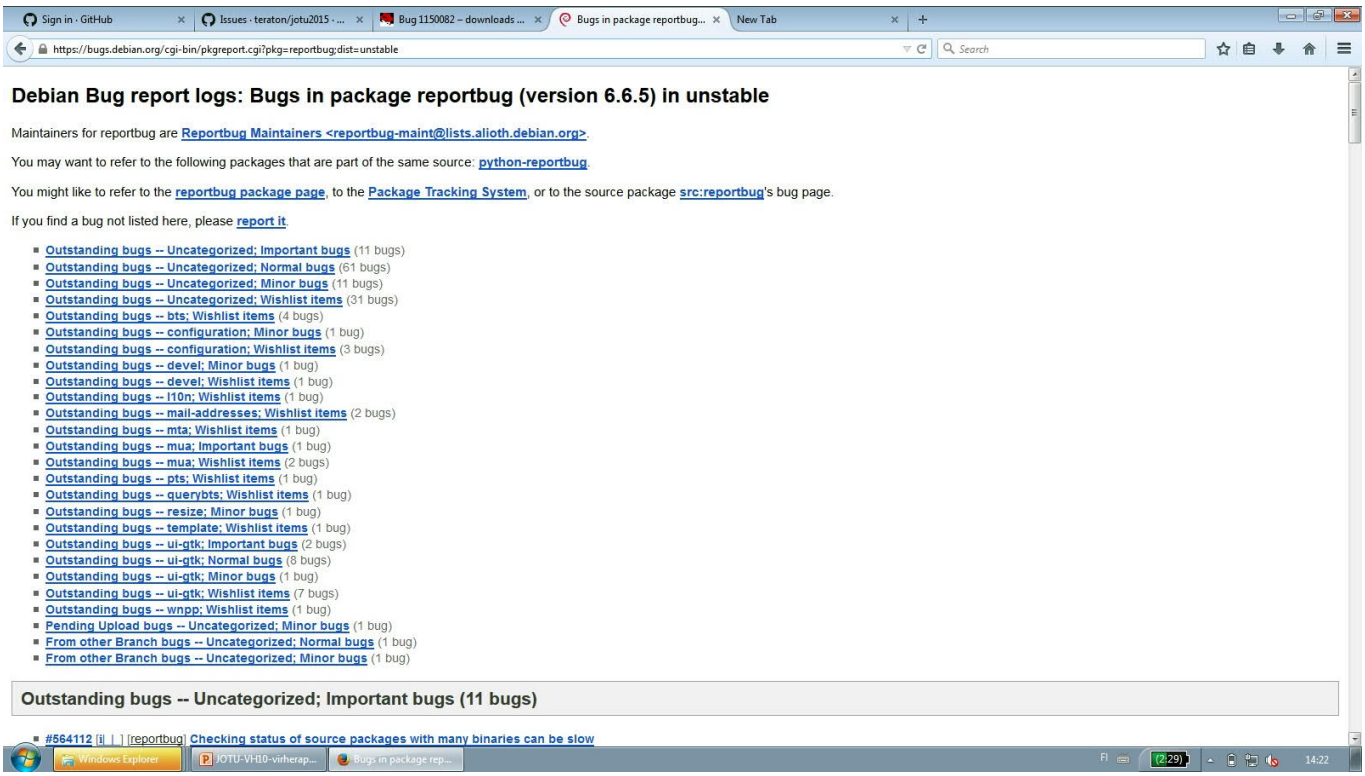


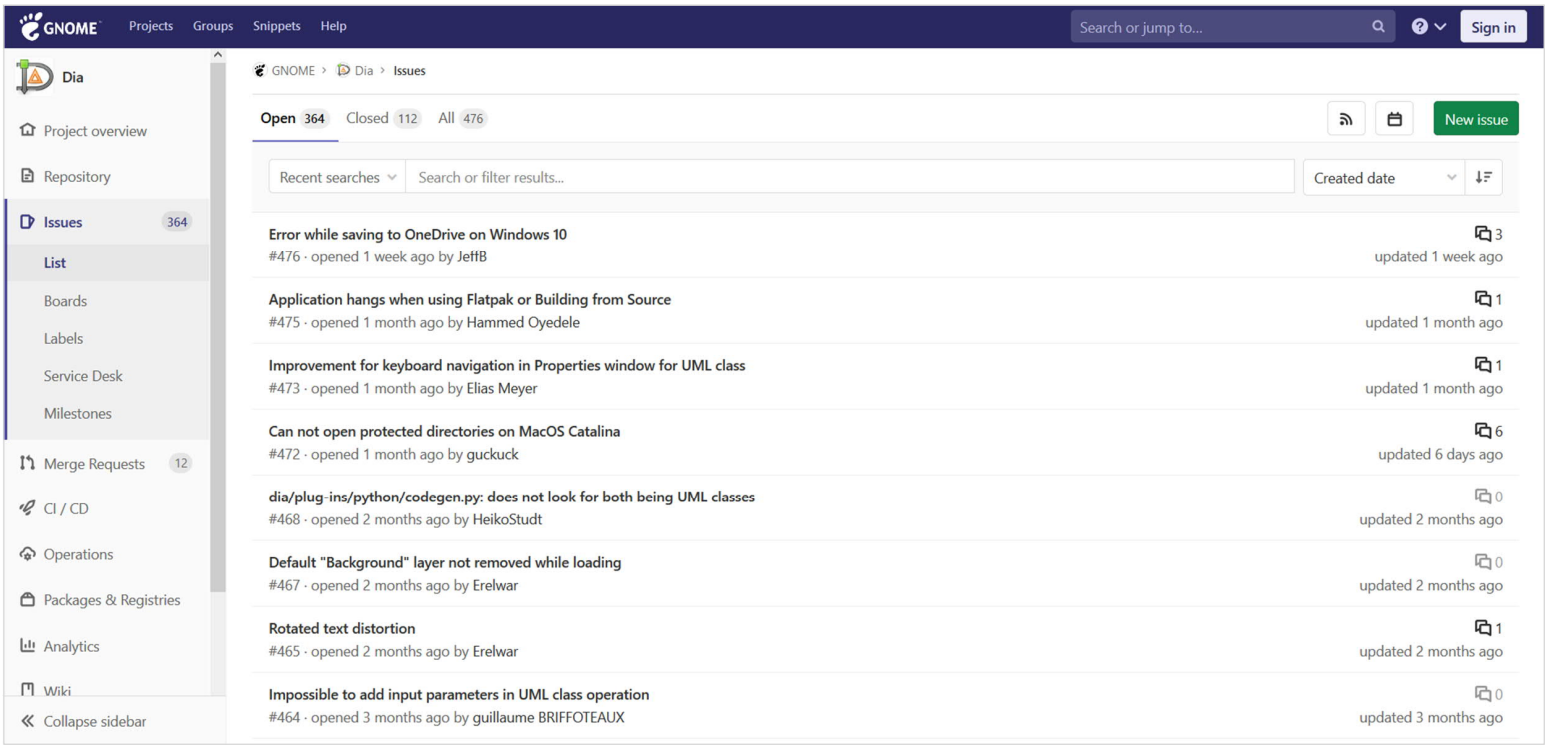
code.google.com/p/chromium/issues/list





bugs.debian.org/cgi-bin/pkgreport.cgi?pkg=reportbug





TDD = test driven development

http://agiledata.org/essays/tdd.html

Test-driven development (TDD) (Beck 2003; Astels 2003), is an evolutionary approach to development which combines test-first development (TFD) where you write a test before you write just enough production code to fulfill that test and refactoring.

What is the primary goal of TDD? One view is the goal of TDD is specification and not validation (Martin, Newkirk, and Kess 2003). In other words, it's one way to think through your requirements or design before your write your functional code (implying that TDD is both an important agile requirements and agile design technique). Another view is that TDD is a programming technique. As Ron Jeffries likes to say, the goal of TDD is to write clean code that works. I think that there is merit in both arguments, although I lean towards the specification view, but I leave it for you to decide.

https://www.guru99.com/test-driven-development.html

TEST DRIVEN DEVELOPMENT (TDD) approach first, the test is developed which specifies and validates what the code will do. In simple terms, test cases are created before code is written. The purpose of TDD is to make the code clearer, simple and bug-free.

Test-Driven Development starts with designing and developing tests for every small functionality of an application. TDD instructs developers to write new code only if an automated test has failed. This avoids duplication of code. The full form of TDD is Test-driven development.

The simple concept of TDD is to write and correct the failed tests before writing new code (before development). This helps to avoid duplication of code as we write a small amount of code at a time in order to pass tests. (Tests are nothing but requirement conditions that we need to test to fulfill them).

Test-Driven development is a process of developing and running automated test before actual development of the application. Hence, TDD sometimes also called as Test First Development.

https://www.freecodecamp.org/news/test-driven-development-what-it-is-and-what-it-is-not-41fa6bca02a2/

Many programmers have tried this TDD technique, failed, and concluded that TDD is not worth the effort it requires. Some programmers think that, in theory, it is a good practice, but that there is never enough time to really use TDD. And others think that it is basically a waste of time.

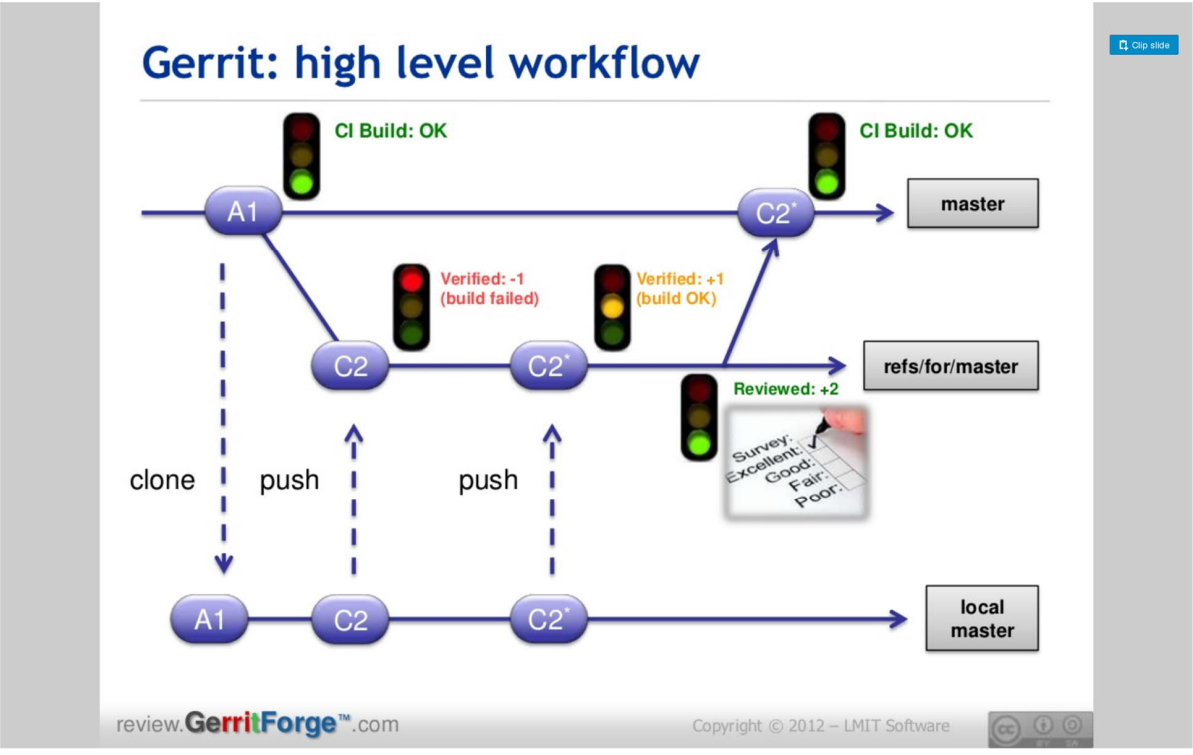
There is a very good book on TDD, Test Driven Development: By Example, by Kent Beck, if you want to check it out and learn more.

Uncle Bob describes TDD with three rules:

- You are not allowed to write any production code unless it is to make a failing unit test pass.
- You are not allowed to write any more of a unit test than is sufficient to fail; and compilation failures are failures.
- You are not allowed to write any more production code than is sufficient to pass the one failing unit test.

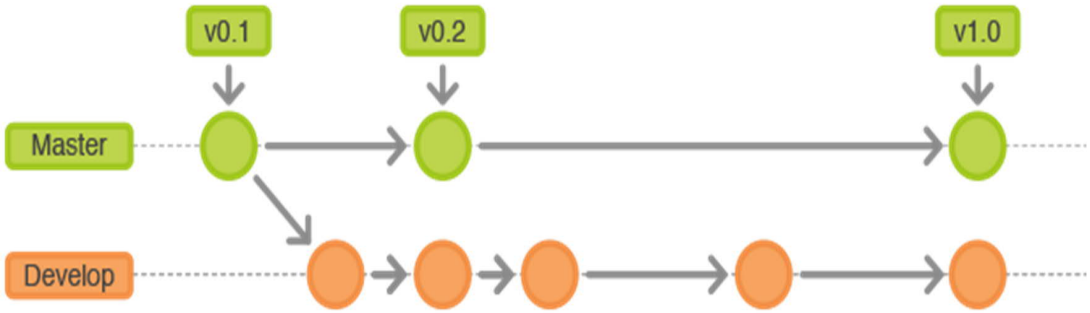
Version control

Tampereen yliopisto
Tampere University



Gitflow Workflow: Historical branches

- Two branches are used to record project history
- Master branch → stores the official release history
- Development branch → Used for integrating all features



Gitflow Workflow: Feature branches

- Feature branches
  - Each new feature should reside in its own branch
  - Feature branches use develop as their parent branch.
  - Features should never interact directly with master.

Gitflow Workflow: Release branches

- Release branches
  - Once develop has acquired enough features for a release, you fork a release branch off of develop
  - Prepare for release

Gitflow Workflow: Hot fix branches

- Hot fix branches
  - to quickly patch production releases.
  - You can think of them as ad-hoc release branches that work directly with master

Baselines

- Baselines may be specified using a configuration language, which allows you to **define what components are included in a version** of a particular system.
- Baselines are important because you often have to recreate a specific version of a complete system.
  - For example, a product line may be instantiated so that there are individual system versions for different customers. You may have to recreate the version delivered to a specific customer if, for example, that customer reports bugs in their system that have to be repaired.

Agile building

- Check out the mainline system from the version management system into the developer's private workspace.
- Build the system and run **automated tests** to ensure that the built system passes all tests. If not, the build is broken and you should inform whoever checked in the last baseline system. They are responsible for repairing the problem.
- Make the changes to the system components.
- Build the system in the private workspace and rerun system tests. If the tests fail, continue editing.

Agile building

- Once the system has passed its tests, check it into the build system but do not commit it as a new system baseline.
- Build the system on the build server and run the tests. You need to do this in case others have modified components since you checked out the system. If this is the case, check out the components that have failed and edit these so that tests pass on your private workspace.
- If the system passes its tests on the build system, then commit the changes you have made as a new baseline in the system mainline.

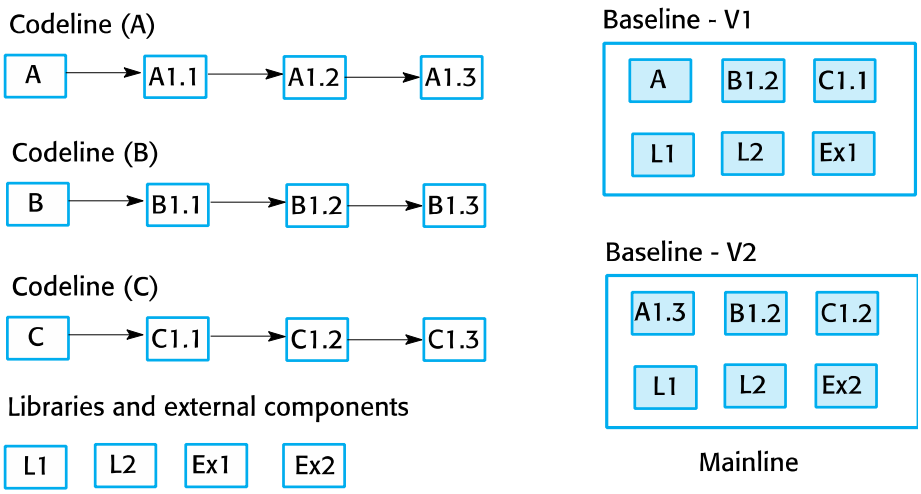
Version management

- Version management (VM) is the process of keeping track of different versions of software components or configuration items and the systems in which these components are used.
- It also involves ensuring that changes made by different developers to these versions do not interfere with each other.
- Therefore version management can be thought of as the process of managing codelines and baselines.

Codelines and baselines

- A codeline is a sequence of versions of source code with later versions in the sequence derived from earlier versions.
- Codelines normally apply to components of systems so that there are different versions of each component.
- A baseline is a definition of a specific system.
- The baseline therefore specifies the component versions that are included in the system plus a specification of the libraries used, configuration files, etc.

Codelines and baselines



Version control systems

- Version control (VC) systems identify, store and control access to the different versions of components. There are two types of modern version control system
  - Centralized systems, where there is a single master repository that maintains all versions of the software components that are being developed. Subversion is a widely used example of a centralized VC system.
  - Distributed systems, where multiple versions of the component repository exist at the same time. Git is a widely-used example of a distributed VC system.

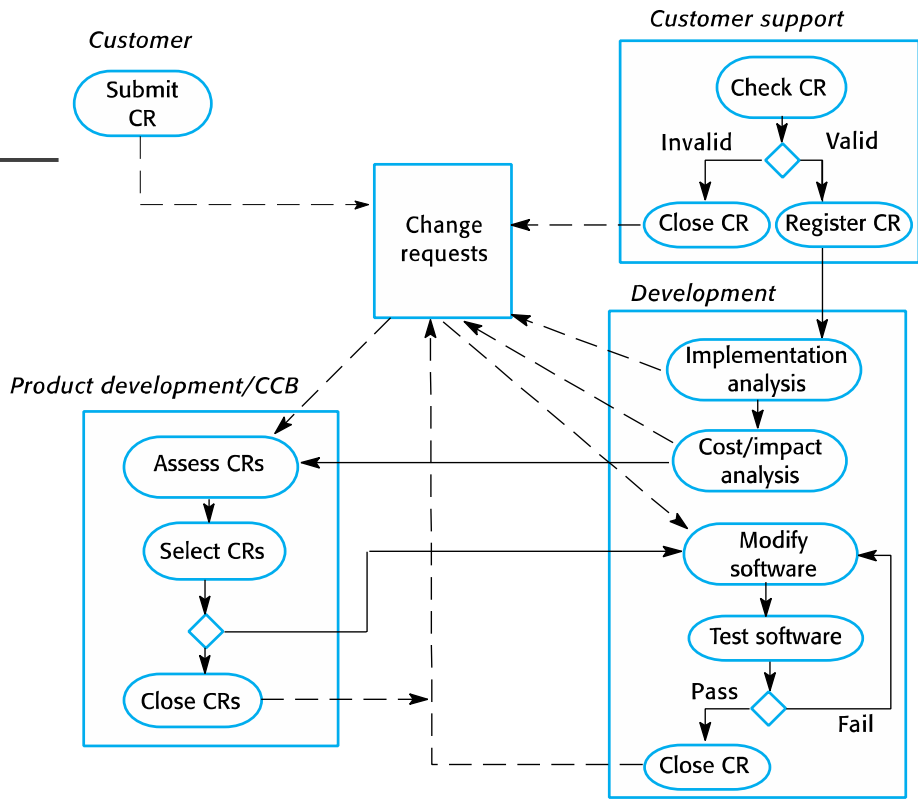
Key features of version control systems

- Version and release identification
- Change history recording
- Support for independent development
- Project support
- Storage management.

Change management

Change management

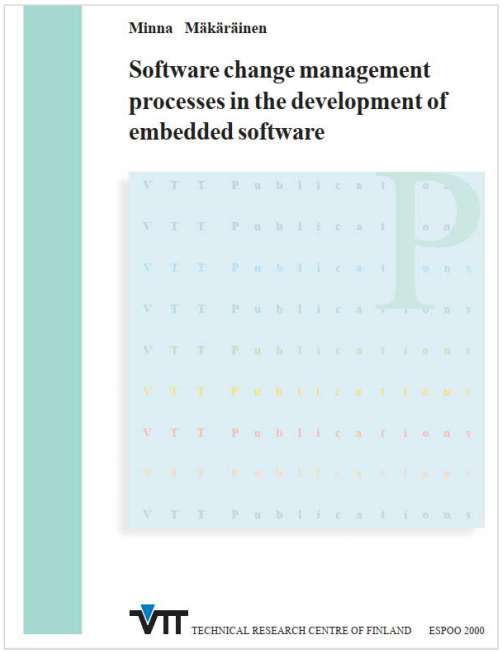
- Organizational needs and requirements change during the lifetime of a system, bugs have to be repaired and systems have to adapt to changes in their environment.
- Change management is intended to ensure that system evolution is a managed process and that priority is given to the most urgent and cost-effective changes.
- The change management process is concerned with analyzing the costs and benefits of proposed changes, approving those changes that are worthwhile and tracking which components in the system have been changed.



The change management process

Factors in change analysis

- ✧ The consequences of not making the change
- ✧ The benefits of the change
- ✧ The number of users affected by the change
- ✧ The costs of making the change
- ✧ The product release cycle



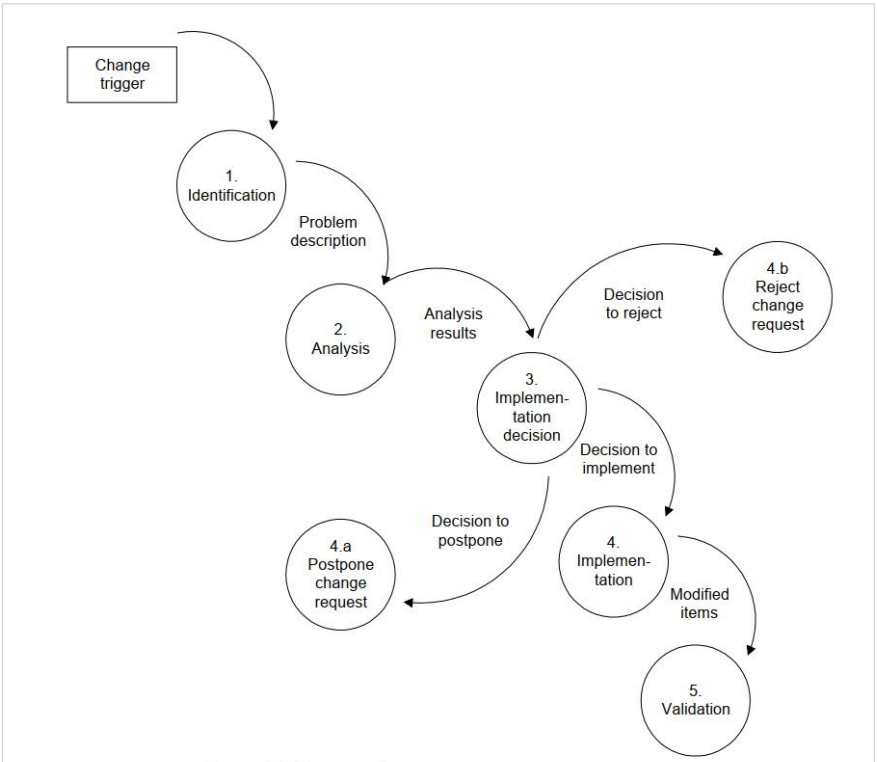
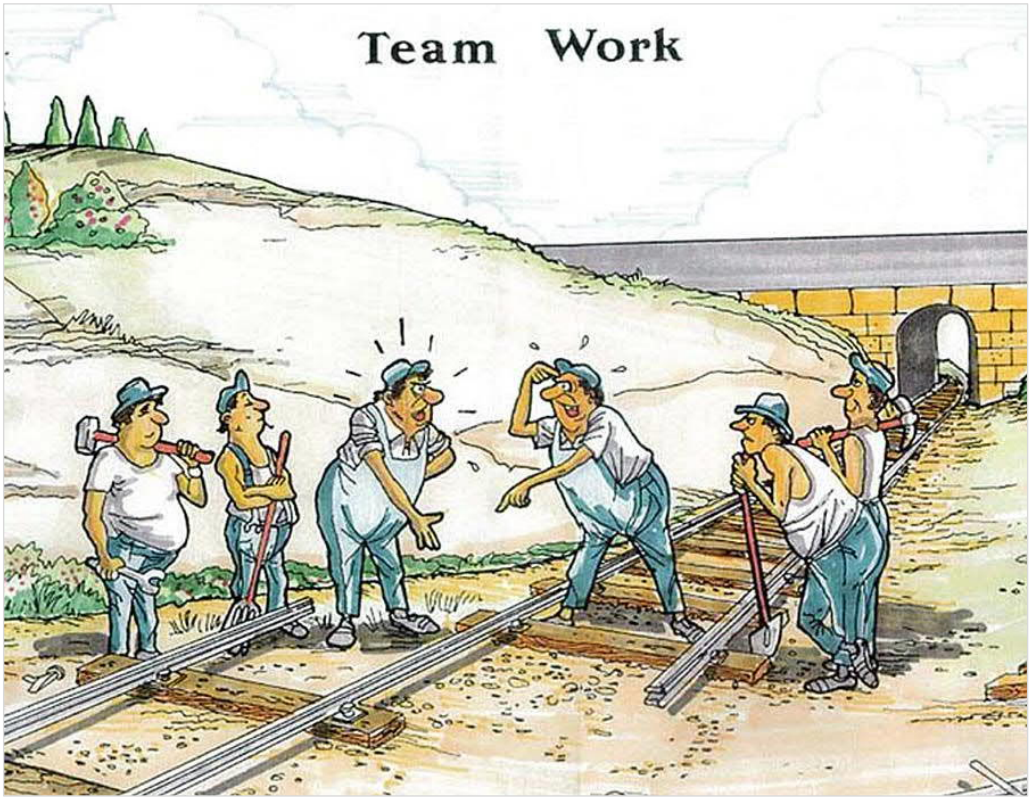


Figure 14. Generic change management process.



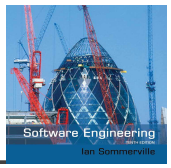
Tampereen yliopisto
Tampere University

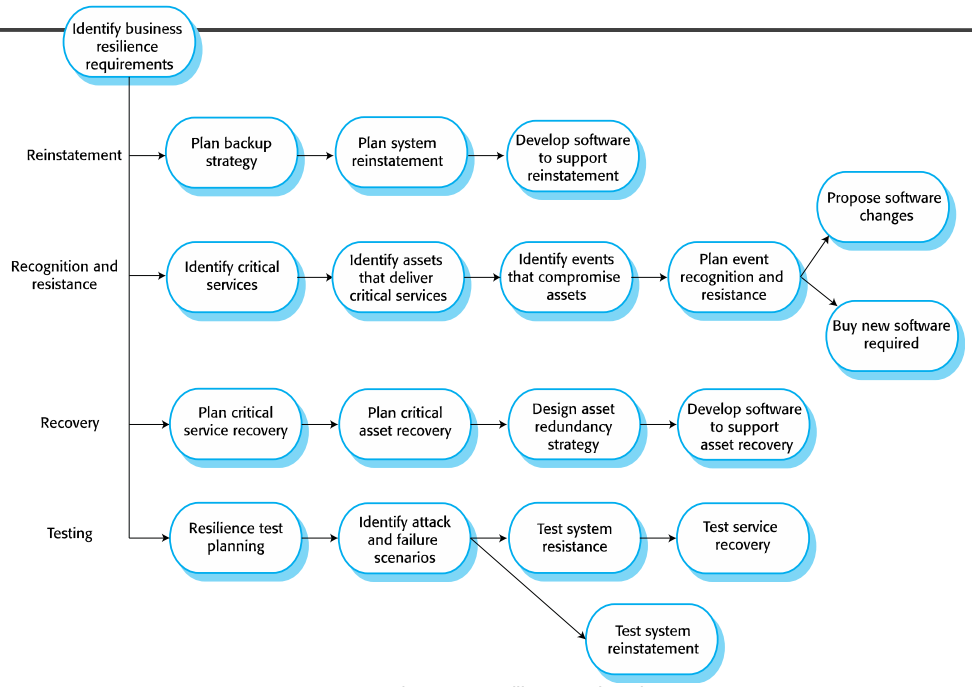
V & V = verification and validation

Tampereen yliopisto
Tampere University

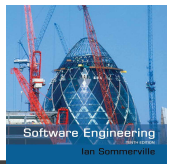
Resilience

Resilience engineering





Streams of work in resilience engineering



- ✧ Identify business resilience requirements
- ✧ Plan how to reinstate systems to their normal operating state
- ✧ Identify system failures and cyberattacks that can compromise a system
- ✧ Plan how to recover critical services quickly after damage or a cyberattack
- ✧ Test all aspects of resilience planning

CM = configuration management

Tampereen yliopisto
Tampere University